Debreceni Egyetem

Informatikai Kar

Forráskód hasonlóság

Témavezető
Noszály Csaba
egyetemi tanársegéd

Készítette:
Pancsira Gábor
Programtervező informatikus (Bsc)

Debrecen
2011

Tartalomjegyzék

I. Bevezetés

Szakdolgozatom témája az informatikai körökben igen kényes témaként kezelt forráskód hasonlóság, pontosabban azok a szoftverek, melyeket annak detektálására készítettek el.

A mindennapi életben plágiumnak vagy plagizálásnak nevezik azt a cselekedetet, ha valaki egy másik ember (az eredeti szerző) munkáját saját publikált munkájában hivatkozás, forrás megjelölés és/vagy szerzői engedély nélkül felhasználja, azt sajátjaként tünteti fel, és ezzel az eredeti szerző jogait sérti. Informatikai esetben ez azt jelenti, hogy egy személy olyan forráskódot – vagy forráskód részletet – próbál sajátjaként feltűntetve felhasználni, amit valójában egy másik személy készített. Informatikai esetben azért nehezebb egy konkrét plágium esetet megállapítani, mert bevett szokás egy adott probléma megoldása oly módon, amit nem saját magunk fejlesztettünk ki. A plágiumdetektáló rendszerek használatosak mind akadémiai, mind üzleti körökben. Ugyanakkor a plágium legtöbb esetben akadémiai körülmények között fordul elő, ezért szakdolgozatom során is ebből az aspektusból fogom vizsgálni a plágiumdetektálás folyamatát. A diákok – szándékosan vagy véletlenül – használnak fel forrásokat munkáikban, anélkül hogy hivatkoznának annak eredeti szerzőjére. A plágium manuális észlelése több tíz vagy több száz hallgatói munkában történő végrehajtása korábban is rendkívül időigényes és gazdaságtalan volt, és az informatikus hallgatók számának növekedése mostanra teljesen lehetetlenné tette azt. Ezért különböző szoftveres eszközök készültek, melyek segítséget nyújtanak a plágium jelenlétének meghatározásában.

A szakdolgozatomban több plágiumdetektáló szoftvert szeretnék ismertetni a teljesség igénye nélkül azok funkciói alapján, majd pedig azok tesztelése is sorra kerül, mely során vizsgálni fogom azok teljesítményét és a különböző plágium elrejtő módszerekkel szembeni érzékenységét.

A rendszerek ismertetésénél nem fogok kitérni azok mély működésére, inkább be szeretném mutatni az azokban rejlő lehetőségeket, azok használhatóságát.

Célom a szakdolgozat készítése során nagyobb rálátást szerezni a plágiumdetektáló szoftverek által nyújtott segítség mértékére és megbízhatóságára, valamint a másik oldalról figyelve a plágium elrejtésének lehetőségeire. Ezen kívül szeretném bemutatni néhány

rendszer használatát is: az egyes rendszerekkel használata során esetlegesen felmerülő problémákat és az azok által nyújtott segítséget egyaránt. Mindazonáltal ki kell jelentenünk, hogy az szakdolgozatomban említett rendszerek egyike sem képes bizonyítékot szolgáltatni a plágium tényére.

II. Az alapok

Tipikusan ezek a szoftverek a programpárokra egy hasonlósági értéket adnak vissza. Ezen érték alapján döntheti el a plágiumot kereső személy, hogy ez a hasonlóság plágium által okozott, vagy a hasonlóság azért áll fent, mert az adott probléma implementálása egy általános módon történt (például általában a listákat ugyanolyan módon implementálják azok a diákok, akik ugyanazon oktatónál tanultak). Ebből kifolyólag szokás a plágium-gyanús programok készítőit is meghallgatni arról, hogy milyen módon jutottak arra a megoldásra, amit implementáltak.

Először is nézzük meg, hogy a különböző eszközök milyen megvalósítási stratégiát alkalmaznak a plágium detektálás során. A legtöbb ilyen eszköz azzal kezdi a vizsgálatát, hogy az eredeti forráskódból egy úgynevezett „token stringet” készít, melyben az egyes tokenek azok a lexikális egységek, melyekből a program egyes blokkjai felépülnek (ilyenek a azonosítók, kulcsszavak stb.). A folyamat során a forráskód azon részei, melyek megváltoztatása esetén a program szemantikája nem változik (kommentek, változónevek, formázási elemek) eltávolításra kerülnek. Ezután a forráskódok token string reprezentációi kerülnek összehasonlításra, a hasonlóság megállapításának érdekében.

A korai rendszerek egy tisztán attribútum alapú megközelítést használtak: a forráskódok különböző tulajdonságait számolták meg, ezáltal az egyes programokat egy-egy számsorozattal reprezentálva. Ezeknél a rendszereknél plágium gyanús esetről akkor beszéltek, ha a két fájlt reprezentáló számsorok ’eléggé hasonlóak voltak’.

Később struktúra alapú rendszereket dolgoztak ki, melyek ahelyett, hogy az egyes programokat számsorokká alakítanák, azok hasonlóságát a struktúrájuk összehasonlításával mérték. Általában az eredeti programokat tokenekre bontják, és az ez által kapott token stringeket hasonlítják össze különböző string összehasonlító algoritmusokkal (például Running-Karp-Rabin, Greedy String Tiling). Struktúra alapúnak tekinthetjük azokat a rendszereket is, melyek program függőségi gráfot alkalmaznak.

A forráskód hasonlóság detektáló programok hasonlítását alapvetően két részre bonthatjuk: funkció szerinti összehasonlítás, illetve teljesítmény alapú összehasonlítás.

III. Az eszközök funkció szerinti összehasonlítása

A funkció szerinti összehasonlítás minőségi összehasonlítás, mely többek között kitér arra, hogy az egyes eszközök milyen tulajdonságokkal rendelkeznek, milyen programozási nyelveket támogatnak, helyi vagy webes alapokon működnek, milyen algoritmust használnak. Természeténél fogva az ilyen összehasonlítás pusztán leíró jellegű, az ilyen alapokon történő összehasonlításra támaszkodva nagyon nehezen állapítható meg, melyik eszköz a legjobb.

Szakdolgozatomban az egyes forráskód hasonlóságot vizsgáló rendszerekről az alábbi, minőséget leíró tulajdonságokat tárgyalom:

- Támogatott nyelvek: A rendszer által támogatott programozási nyelvek.
- Kiterjeszthetőség: Lehetséges-e a támogatott nyelvek körének bővítése valamilyen bővítmény vagy konfigurálás segítségével? Ez a számos használható nyelv miatt nagyon hasznos lehet, hogy ne legyen szükség esetlegesen másik plágiumdetektáló szoftverre, ha egy vizsgálandó dokumentum olyan nyelven íródott, amelynek vizsgálatára a rendszerünk alapértelmezésben nincs felkészítve.
- Az eredmények prezentációja: az eszköz használata után rengeteg erőfeszítést igényel annak megállapítása, hogy az esetlegesen talált hasonlóságok valós plágium esetek, vagy puszta véletlenek. Ennek a procedúrának az időigénye általában többszöröse a gépi vizsgálattal eltöltött időnek. Ahhoz hogy ez az időt minimálisra csökkenjen, fontos az eredmények lényegre törő, jól átlátható prezentálása.

  Egy jól elkészített jelentés az eredményekről mindenképpen tartalmazza az alábbiakat:

  Összegzés: Itt érdemes szerepeltetni az összes vizsgált beadvány, a sikeres elemzések számát, a vizsgálat futásához megadott paramétereket, valamint egy táblázatot, amelyben a hasonlóságok eloszlása jelenik meg. Ezen kívül szerepelhet egy hisztogram is, amely segít meghatározni azokat a hasonlósági értékeket, amelyeket mindenképpen érdemes tovább vizsgálni plágium szempontjából.

  Egyezések: Az egyezéseket hasonlóság szerint rendezve, átfogóan kell listázni. Ez történhet akár páronként, akár csoportokban. A kívánt erőfeszítések csökkentésének érdekében praktikus megoldás, ha megadható egy hasonlósági

küszöb, ami alatt nem szerepelnek a csoportban a további beadványok – mivel azok nem plágium gyanúsak.

Forráskód összehasonlító eszköz: Annak érdekében, hogy a hasonlónak tűnő forráskódokat könnyedén összehasonlíthatóak legyenek, szintén nagy segítség egy beépített szerkesztő, amely képes mindkét forráskódot egy időben a képernyőn megjeleníteni.

- Használhatóság: Egy jó eszköz használata legyen egyszerű és gyorsan tanulható. Ebből a szempontból például egy grafikus felhasználói felület sokkal hasznosabb, mint egy parancssoros felület.
- Sablon kódok kizárása: Teljesen természetes, hogy egyes programozási problémákat a programozók sablonok alapján oldanak meg, például rendezéseket, adatszerkezeteket sablon alapján implementálnak. Rendszeresen előfordul az is, hogy nem egy teljes programkód készítése a feladat, hanem valamilyen már meglévő, de nem teljes alapkódot kell kiegészíteni a probléma megoldásához. Ezekben az esetekben ezek a megegyező kódrészletek plágium gyanúját vethetik fel. Egyes rendszerek megengedik, hogy a felhasználó az ilyen közös kódokat elhelyezzék egy bázisfájlba, és ez által kizárják azokat a hasonlóságkeresési fázisból. Ezáltal sok fals pozitív eredmény szűrhető ki.
- Kisméretű fájlok kizárása: a sablon kódok kizárásához kapcsolódik a kisméretű fájlok kizárása. Ilyen nagyon kicsi fájlokról beszélhetünk például a Java beanek esetében, amely fájlok mindössze attribútumokból, és az azokhoz tartozó get és set metódusokbl áll. Ezek a fájlok is okozhatnak fals pozitív eredményeket, hiszen rengeteg programozó használ ugyanolyan osztályimplementálási módszereket. Ezt az eszköz jelezheti az eredményeknél a fájl méretével, vagy lehetőséget nyújthat egy bizonyos – a felhasználó által megszabott – méretnél kisebb fájlok kihagyására a vizsgálatból.
- Evolúciós összehasonlítás: Ezzel a kritériummal az a kérdés vetődik fel, hogy az adott eszköz képes-e különböző verziókat megkülönböztetni az ugyanazon feladathoz tartozó megoldásokra, anélkül hogy azokat kölcsönösen újra összehasonlítaná a régebbi verziókkal. Ez lehetséges a különböző verziók különböző könyvtárakba helyezésével, vagy azáltal, hogy az eszköz indításakor

lehetőség van annak megadására, hogy melyik kód melyiknek régi vagy új verziója.

- Beadvány vagy fájl alapú értékelés: Attól függően, hogy egy eszköz fájlonként, vagy beadványonként végzi el az értékelést, változhat az eredmények prezentációjának használhatósága. Amikor egy beadvány több fájlból áll, fontos hogy a plágium vizsgálat minden egyes fájlon megtörténjen, hiszen ha egy több fájlból álló beadványt csak egy fájl alapján értékel az eszköz, akkor könnyedén plágium gyanúsnak tűntetheti fel az egész beadványt – akár hamisan is. Ettől függetlenül a beadványok összehasonlítása történhet a fájlok alapján, de ez felveti a kérdést, hogy az egyes fájlok hasonlósági eredményének milyen módon történő kombinálásával állítódik elő az egész beadványra vonatkozó hasonlósági pontszám.
- Helyi vagy web alapú: A web alapú eszközöknél az interneten történő kommunikáció magában hordozza annak a kockázatát, hogy a kommunikáció során bizalmas információkat fedünk fel a külvilág számára.
- Nyílt forráskódú: A nyílt forráskódú rendszerek előnye, hogy adott a lehetőség a rendszert úgy alakítani, hogy az jobban megfeleljen az adott környezetben tapasztalható igényeknek, felhasználási céloknak.

Az alábbiakban öt kiválasztott rendszert fogok bemutatni, az előbb felsorolt szempontok alapján jellemezve őket. Ezek a rendszerek a következők: JPlag, Marble, MOSS, Plaggie, SIM.

a. JPlag

A JPlagot Guido Malpohl fejlesztette ki a Karlsruhei Egyetemen. A fejlesztés 1996-ban még egy hallgatói kutató projektként indult, pár hónappal később pedig már az első online tesztelő rendszerként működött. 2005-ben változtatta webes szolgáltatássá Emeric Kwemou és Moritz Kroll.

A JPlag a forráskódot token stringekké alakítja, melyek a program struktúráját reprezentálják, tehát mondhatjuk, hogy a JPlag struktúra alapú megközelítéssel dolgozik. Két

token string összehasonlítására a rendszer a Greedy String Tiling algoritmust alkalmazza, bizonyos optimizálásokkal kiegészítve a jobb hatásfok érdekében.

- Támogatott nyelvek: A JPlag támogatja a Java, C#, C, C++ és a Scheme programozási nyelveket, valamint a természetes nyelven íródott szövegeket.
- Kiterjeszthetőség: A JPlag csak addig függ a forráskód nyelvétől, amíg azt token stringgé nem alakítja. Ezt a részét a JPlagnak front-endnek nevezik. Arról nem található leírás, hogyan készíthető front-end olyan nyelvhez, amit a JPlag jelenleg nem támogat.
- Az eredmények prezentációja: A JPlag az eredményeit HTML oldalak formájában tárja elénk. Ezeket az oldalakat a kliensnek visszaküldi, azok a kliensen tárolódnak. A főoldal egy áttekintő oldal, melyen szerepel egy táblázat, ami a vizsgálat során használt konfigurációt tartalmazza, egy lista a meghiúsult elemzésekről, egy diagram a hasonlósági értékek eloszlásáról, valamint egy rendezett lista a legnagyobb hasonlóságot mutató párokról.

  A JPlag egyik jellegzetes funkciója a párok csoportosítása, mely könnyen láthatóvá teszi, ha egy beadvány több másik beadványhoz is hasonlóságot mutat. Amikor kiválasztunk két beadványt, azok fájljai egymás melletti keretekben jelennek meg, színesen kiemelve azokat a részeket, amelyek hasonlónak találtattak a JPlag által. A képernyő tetején lista formájában megtalálható az összes hasonlóságot mutató rész, melyeket kiválasztva a szerkesztő automatikusan a megfelelő kódrészlethez ugrik.
- Használhatóság: A könnyen használható Java Web Start kliens, az eredmények rendezett, jól átlátható listázása és a kódok összehasonlítását elősegítő beépített szerkesztőnek köszönhetően a JPlag egy nagyon könnyen használható alkalmazás.
- Sablon kódok kizárása: Az elemzés megadható egy báziskódokat tartalmazó könyvtár, mely fájlok formájában tartalmazza a kizárni kívánt kódokat.
- Kisméretű fájlok kizárása: Igen.
- Evolúciós összehasonlítás: Nem.
- Beadvány vagy fájl alapú értékelés: A JPlag feltételezi, hogy minden programhoz, amelyet meg akarunk vizsgálni, saját könyvtár tartozik. Az egyazon mappában található fájlokkal a JPlag mint egy beadvány dolgozik. Az értékelés beadvány alapú.

- Helyi vagy web alapú: A JPlag mint web alapú szolgáltatás áll rendelkezésünkre. A JPlag weboldalán elérhető egy Java Web Start kliens, melynek segítségével feltölthetjük a vizsgálni kívánt fájlokat a szerverre. Ezen kívül egy elég körültekintő leírás is elérhető arról, hogyan írhatjuk meg a saját JPlag kliensünket.
- Nyílt forráskódú: Nem.

b. Marble

A Marble plágiumdetektáló eszközt 2002-ben fejlesztette ki Jurriaan Hage az Utrechti Egyetemen. A cél az volt, hogy egy egyszerű, könnyen karbantartható eszközt készítsen, amit a Java programok közötti gyanús hasonlóságok észlelésére használhatnak. Összegyűjtve, majd összehasonlítva az egyetemen készült összes programot, a Marble több tucat bizonyított plágium esetre derített fényt. A skálázhatóság érdekében fontos, hogy az eszköz különbséget tudjon tenni régi és új verziók között, ezáltal elkerülve hogy egyazon problémára készült új és régi megoldások ne kerüljenek összehasonlításra.

A Marble struktúra alapú megközelítést használ a beadványok vizsgálatánál. A folyamat elején a beadványt az eszköz fájlokra bontja, úgy hogy egy fájl mindössze egy felső osztályt tartalmaz. A következő fázisban ezeket a fájlokat normalizálja, vagyis eltávolítja azokat az elemeket, melyek könnyen megváltoztathatóak. Ez a folyamat egy lexikai elemzéssel kezdődik, mely megőrzi a kulcsszavakat és a gyakran használt osztály és metódusneveket. Ezután a kommentek, a túlzott üres területek, szöveges konstansok és az import deklarációk eltávolításra kerülnek, az egyéb egységeket pedig helyettesítik a típusuknak megfelelő jelzéssel. Például minden hexadecimális számot H-val, minden karakteres literált pedig L-lel.

Az eszközt használó személy választhat, hogy az osztályozott vagy az osztályozatlan normalizált verziókat akarja összehasonlítani, esetleg mindkettőt. Mivel az osztályozás heurisztikus, egy apró változtatás valamely osztály metódusaiban teljes mértékben meg tudja változtatni azt. Készült megfigyelés olyan esetről, amikor egy hallgató számos változtatást eszközölt a metódusokban, de azok sorrendjét a forrásban nem változtatta meg, viszont a vizsgálat eredményét – negatív irányba – megváltoztatta. Ezért érdemes az osztályozatlan verziókat is összehasonlítani.

A normalizált fájlok tényleges összehasonlítása a diff nevű Unix/Linux segédprogrammal történik. Ezután a pontokat a különböző sorok és összes sorok arányából számolják.

- Támogatott nyelvek: A Marble támogatja a Java nyelvet. Kisérleti státuszban van a Perl, PHP és XSLT támogatás.
- Kiterjeszthetőség: A nyelvfüggő rész a normalizációs fázis, amely könnyedén igazítható a hasonló programozási nyelvekhez.
- Eredmények prezentációja: Az eredményeket szkriptek formájában kapjuk meg: az osztályozatlan fájlok hasonlításának eredménye a suspects.nf, az osztályozottak hasonlításának eredménye a suspects.nfs nevű fájlokban tárolódik. Ezeket ha lefuttatjuk, kiíratásra kerülnek azon párok hasonlósági pontszáma és mérete, melyek meghaladják az előzetesen megszabott hasonlósági küszöböt, majd mindkét eredeti fájlt megnyitja egy diff szerkesztőben, hogy megtekinthessük a különbséget a kettő között. A felhasználónak ugyanakkor lehetősége van arra, hogy a szkripteket futtatás helyett egy szövegszerkesztőben megnyitva saját maga vizsgálja meg a gyanús eseteket.
- Használhatóság: A Marble elérhető egy Perl szkript formájában és karakteres kezelőfelülettel rendelkezik.
- Sablon kódok kizárása: Nincs lehetőség sablon kódok kizárására.
- Kisméretű fájlok kizárása: A Marble szkript egy küszöb értéket használ a bizonyos méretnél kisebb fájlok vizsgálatból történő kizárására.
- Evolúciós összehasonlítás: Amennyiben a beadványokat egy megfelelően rendszerezve tároljuk, akkor a Marble képes az új beadvány minden fájlját összehasonlítani bármely másik beadvánnyal, anélkül hogy a régebbi életciklusban készülteket egymással összehasonlítaná. Megfelelően rendszerezett tárolás: minden könyvtár egy probléma megoldásait tartalmazza, a különböző életciklusoknak különböző alkönyvtárat biztosítva, melyeken belül a különböző verziók kapnak különböző alkönyvtárakat.
- Beadvány vagy fájl alapú értékelés: Minden egyes fájlpárhoz rendelődik egy hasonlósági pontszám.
- Helyi vagy web alapú: A Marble futása a kliensen történik.
- Nyílt forráskódú: Nem.

c. Measure Of Software Similarity (MOSS)

A MOSS-t 1994-ben fejlesztették ki a Stanford Egyetemen. Webszolgáltatásként működik, mely elérhető egy, a MOSS honlapján keresztül hozzáférhető szkript segítségével, mely Unix/Linux rendszerek alatt működik.

A dokumentumok hasonlóságának méréséhez a MOSS az adott dokumentumok szabványosított verzióját hasonlítja össze: a MOSS a winnowing nevű dokumentum-lenyomatkészítő algoritmust használja. A dokumentum-lenyomatkészítés egy olyan módszer, ami a dokumentumot határos alsztringekre bontja, majd minden egyes alsztringet hashel. Ezeknek a hasheknek a részhalmazát a dokumentum lenyomatának nevezzük. A winnowing egy hatékony algoritmus ezeknek a részhalmazoknak a kiválasztására.

- Támogatott nyelvek: A MOSS jelenleg a következő nyelven íródott kódokat tudja elemezni: C, C++, C#, Java, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, HCL2.
- Kiterjeszthetőség: A MOSS kiterjeszthető, a kiterjesztést a fejlesztők végzik.
- Eredmények prezentációja: A MOSS kimenete egy HTML prezentáció linkekkel és egy HTML alapú diff szerkesztővel, mellyel könnyedén navigálhatunk az eredmények között. Az eredmény a MOSS web szerverén tárolódik, elérése interneten keresztül történik. Az eléréshez szükséges linket a dokumentumok vizsgálatának befejezése után kapjuk meg. Annak érdekében, hogy az eredmények privátak maradjanak, több óvintézkedés is történt: azok hozzáférése nem lehetséges sem az internetezők emberek, sem robotok számára. Ezt az URL-be ágyazott véletlenszerűen generált számmal biztosítják, mely számról mindössze a vizsgálandó fájlokat beadó felhasználó értesül. Az eredmények elérése kizárólagosan ezzel az URL-lel történhet. Az eredményeket tartalmazó oldalak automatikusan elévülnek 14 nap után, és a forráskódok sem tárolódnak a szerveren ennél tovább. Ezek az intézkedések együttvéve egy lehetséges visszaélés esélyeit a minimálisra csökkentik.
- Használhatóság: A MOSS rendszerébe történő regisztrálás után e-mailben egy beadó szkriptet kapunk, melynek használatával tudjuk feltölteni a MOSS szerverére a vizsgálni kívánt dokumentumainkat.

- Sablon kódok kizárása: A MOSS lehetőséget biztosít arra, hogy megadjunk egy fájlt ami tartalmazza a figyelmen kívül hagyni kívánt kódokat: a MOSS ezeket a kódokat mindenféle vizsgálatból kihagyja. Továbbá a MOSS képes a megosztott kódok kezelésére is, kihagyva a gyanús beadványok közül azokat melyek ilyen kódot tartalmaznak. Ezáltal megelőzve a legális kódmegosztásból adandó fals pozitív eredményeket.
- Kisméretű fájlok kizárása: Igen.
- Evolúciós összehasonlítás: Igen, a felhasználó a parancssoron megadhatja, hogy melyik beadványok ugyanazon probléma megoldásának új vagy régi verziója.
- Beadvány vagy fájl alapú értékelés: Alapértelmezés szerint a MOSS a fájlokat beadványok vagy könyvtárok szerint hasonlítja össze, de a beadási szkript tartalmaz egy opciót, mely által a felhasználó számára megengedett a fájl alapú összehasonlítás.
- Helyi vagy web alapú: A MOSS egy web alapú rendszer.
- Nyílt forráskód: Nem.

d. Plaggie

A Plaggie egy kifejezetten Java nyelvre készült forráskód hasonlóság detektáló eszköz. Megjelenésében és funkcionalitásában hasonlít a JPlagra, de néhány szempontból teljesen más: a Plaggiet a kliensre kell telepíteni, és nyílt forráskódú eszköz. A Plaggiet 2002ben fejlesztették ki a Helsinki- Műszaki Egyetemen. A Plaggie egy független parancssoros Java alkalmazás.

Az alapvető algoritmus, amit két forráskód fájl összehasonlítására alkalmaz ugyan az, mint a JPlagnál: tokenizálás, majd a Greedy String Tiling alkalmazása. A készítők viszont itt nem implementálták a JPlagnál használatos optimalizációkat.

A Plaggie „olvass el” fájljában a szerzők felsorolnak mind ismert sikertelen támadásokat (kommentek, osztálynevek, metódusok, változók megváltoztatása), mind ismert sikeres támadásokat (bizonyos programsorok új metódusba ágyazása, fölösleges programkód alkalmazása, az if-else és case blokkok sorrendjének megváltoztatása), valamint ismert problémákat is (a Plaggie pontossága GUI kód, vagy automatikusan generált kód esetén).

- Támogatott nyelvek: Java 1.5
- Kiterjeszthetőség: Nem.
- Eredmények prezentációja: Alapértelmezésben a Plaggie eredményei egyszerű szöveges formában jelennek meg a szabvány kimeneten és grafikusan tárolásra kerülnek HTML formátumban. Az egyszerű szöveges megjelenítés kikapcsolható. A kimenetbe bele tartozik egy táblázat, amely olyan statisztikai adatokat jelenít meg mint például a különböző hasonlósági értékek eloszlása, a vizsgálat alá vetett fájlok száma, stb. A HTML jelentés tartalmaz egy rendezhető táblázatot ami tartalmazza a „legjobb" eredményeket, és a hozzájuk tartozó hasonlósági értéket. További vizsgálathoz a beadványra kattintva egy osztott képernyőn összehasonlítás céljából megjelenítődnek a fájlok, kiemelve a hasonlóságokat.
- Használhatóság: A Plaggie konfiguráció egy konfigurációs fájlon keresztül történik, ami a beadványokat tartalmazó könyvtárban található. A Plaggie futtatása annak a parancssoros felületével történik.
- Sablon kódok kizárása: A sablon kódokat a felhasználó kizárhatja olyan módon, hogy gondoskodik egy olyan fájlról, mely tartalmazza a kihagyni kívánt sablon kódokat. Ezen felül a Plaggie lehetőséget nyújt bizonyos kódok vizsgálatból történő mellőzésére fájlnév, alkönyvtár név vagy interfész alapján.
- Kisméretű fájlok kizárása: Fájlok méret alapján történő kihagyása nem lehetséges.
- Evolúciós összehasonlítás: Nem.
- Beadvány vagy fájl alapú értékelés: A Plaggie a beadványokat fájlonként hasonlítja össze, de az eredményeket beadványonként számítja.
- Helyi vagy web alapú: A Plaggie helyben, a kliensen fut. A HTML alapú eredményei viszont lehetővé teszik a web alapú publikálást.
- Nyílt forráskódú: Igen, GNU engedéllyel.

e. SIM

A SIM szoftvert 1989-ben fejlesztette ki Dick Grune az Amszterdami VU Egyetemen, a SIM outputját plágiumkeresési riporttá alakító shell szkriptet Matty Huntjens írta. A jelenleg elérhető 2.26-os verzió 2008ban készült.

A SIM a hasonlóságkeresési folyamat során először a forráskódot tokenizálja, majd felépít egy referencia táblát, amit arra használ, hogy megtalálja az egyezéseket a beadott fájlok és a mintafájl között. A megfelelő programozás garantálja, hogy az ehhez felhasznált számítási idő ésszerű határokon belül maradjon.

Bár a SIM karbantartása és támogatása már megszűnt, a forráskódja és néhány gépi kód szabadon elérhető. A készítők szerint a csomag plágium keresési oldala az Amszerdami VU Egyetemhez testreszabott módon készült, ezáltal nem igazán átvehető.

- Támogatott nyelvek: C, Java, Pascal, Modula-2, Lisp, Miranda és természetes nyelvű szöveg.
- Kiterjeszthetőség: A SIM könnyedén bővíthető azáltal, hogy megadunk egy leírást az új nyelvhez tartozó lexikai elemekről.
- Eredmény prezentációja: A SIM az eredményeket egy hagyományos szöveges dokumentum formájában adja meg, ami általános információval szolgál az összehasonlított fájlokról, mint például a tokenek száma az egyes fájlokban, a vizsgált fájlok darabszáma, azok neve, mérete, stb.
- Használhatóság: Parancssoros felületen keresztül kezelhető, gyakorlattól függő könnyedséggel
- Sablon kódok kizárása: Nem.
- Kisméretű fájlok kizárása: Nem.
- Evolúciós összehasonlítás: Elérhető, bár a korábbi verziókat egymás között is összehasonlítja minden egyes vizsgálatnál.
- Beadvány vagy fájl alapú értékelés: A SIM fájlonkénti alapon végzi az összehasonlítást, bár csak akkor történik bármilyen összehasonlítás, ha az összehasonlítandó fájlok különböző beadványok könyvtáraiban találhatóak.
- Helyi vagy web alapú: A SIM futtatása helyben, a kliensen történik.
- Nyílt forráskódú: Igen.

Az alábbi táblázatban a könnyebb összehasonlíthatóság érdekében összegzem a fentebb leírtakat. A támogatott nyelvek ismételten történő felsorolása helyett egyszerűen csak a támogatott nyelvek számát említem. Az „eredmények prezentálása” és a „használhatóság” szempontok osztályozásához egy egytől ötig terjedő skálát vezetek be, ahol az egy a gyenge, az öt a nagyon jó minősítést jelenti.

Funkció	JPlag	Marble	MOSS	Plaggie	SIM
Támogatott nyelvek	6	1	23	1	5
Bővíthetőség	nem	nem	nem	nem	igen
Eredmények prezentálása	5	3	4	4	2
Használhatóság	5	2	4	3	2
Sablon kódok kizárása	igen	nem	igen	igen	nem
Kis méretű fájlok kihagyása	igen	igen	igen	nem	nem
Evolúciós összehasonlítás	nem	igen	nem	nem	igen
Beadvány vagy fájl alapú értékelés	beadvány	fájl	beadvány	beadvány	fájl
Helyi vagy web alapú	web	helyi	web	helyi	helyi
Nyílt forráskódú	nem	nem	nem	igen	igen

Első táblázat: A plágiumdetektáló szoftverek összehasonlítása funkció szerint

IV. Az eszközök teljesítmény alapú összehasonlítása

A teljesítmény alapú összehasonlítás mennyiségi összehasonlítás, általában valamilyen eszközön végrehajtott kísérlet eredményeit írja le, ezáltal sokkal egzaktabb képet adva arról, hogy melyik rendszer a jobb.

a. Érzékenységi vizsgálat

A következő kísérletek a különböző rendszerek gyenge pontjaira és erősségeire szeretnének fényt deríteni. Mivel korább kísérletek során kiderült, hogy a különböző típusú változtatások kombinálása tovább csökkenti a hasonlósági pontszámot, a kísérlet második felében fény derül arra is, hogy ez milyen hatással lehet az egyes rendszerekre. A kísérlet mindkét fázisában fájlok összehasonlítása történik, nem beadványoké.

Az eszközök érzékenységének vizsgálata egy olyan kísérlettel történik, melyben egy animált Quicksort algoritmus 17 különböző verziója készül el. Ezt a feladatot egy osztott programozási kurzuson kapták a hallgatók az Utrechti Egyetemen. A megoldások öt fájlból állnak: QSortAlgorithm.java, QSortApplet.java, QSortView.java, QSortObserver.java, QSortModel.java. Ezek közül az utóbbi 3 fájl kisméretű, és várhatóan nagyon hasonlóak. Ezen okból a kísérlet során a QSortAlgorithm.java és QSortApplet.java fájlok kerülnek a figyelem középpontjába.

Tizenhét stratégiát vizsgálata történik, melyek mindegyike használható a plágiumvizsgálat megtévesztésére. Ezek a módszerek nem változtatják meg a program szemantikáját, csak annak külső megjelenését. Ezen kívül a változások alkalmazásához nincs szükség a program részletekbe menő ismeretére, és a módosítások alkalmazásához szükséges idő is csekély. A 17 változtatás közül 6 az Eclipse néven ismert fejlesztői környezet beépített funkciói.

i. Érzékenység egyetlen refaktorizálásra

Beállítások

Verzió	Leírás
0	Eredeti Animált Gyorsrendezési feladat (két fájl az ötből)
1	Lefordított kommentek és kisebb változtatás az elrendezésben
2	A metódusok 25%-ának áthelyezése
3	A metódusok 50%-ának áthelyezése
4	A metódusok 100%-ának áthelyezése
5	Az osztály-attribútumok 25%-ának áthelyezése
6	Az osztály-attribútumok 25%-ának áthelyezése
7	Refaktorizált grafikus felhasználói felület kód
8	Megváltoztatott importálások
9	Megváltoztatott színek és szöveg a grafikus felhasználói felületen
10	Minden osztály átnevezése
11	Minden változó átnevezése
12	Eclipse kódtisztítás: hozzáférés változtatás (metódus hozzáféréshez 'this' kulcsszó használata, osztálydeklarálás használata statikus hozzáféréshez
13	Eclipse kódtisztítás: kódstílus (blokkok, zárójelek, módosítók használata)
14	Eclipse – Hash kód és equals függvények generálása
15	Eclipse – Stringek kihelyezése
16	Eclipse – Belső osztályok megszűntetése
17	Eclipse – Get és Set metódusok generálása minden attribútumhoz

Második táblázat: Szemantika megőrző módosítások az eredeti forráskódon

A tesztkódok első verziói nulladik verzióként szerepelnek a kísérlet során – tehát mind a QSortApplet.java, mind a QSortAlgorithm.java fájlnak van nulladik verziója. Ezen fájloknak jön létre 17 különböző verziója, a második táblázatban leírt refaktorizálási módszerek alkalmazásával, az egyes verziókban kizárólag egy típusú módosítást végrehajtva. Ezúton remélhetőleg megbecsülhető az egyes eszközök érzékenysége a különböző módosítási stratégiákra.

Mivel egyes eszközök fájl, míg mások beadvány alapon értékelnek, ezért a némelyik esetben az egyes verzióknak különböző beadványi könyvtárat kellett létrehozni, ezzel biztosítva, hogy a kérdéses eszköz minden egyes párra adott vissza egy pontszámot.

A vizsgálat eredményei

Az érzékenység vizsgálat eredményeit az első és második ábra mutatja. Minden egyes változáson átesett verzióhoz az ábrák hat hasonlósági értéket mutatnak.

Az első értéket egy származtatott hasonlósági érték, melynek előállítása során a Unix ’diff’ nevű eszközének alkalmazása történik közvetlenül a forrásfájlokon. Ennek célja hogy a módosítás forráskódon tett hatásának mennyiségi meghatározása megtörténjen, mintegy kiindulási pontot adva a kísérletnek. Az érték kiszámítása a következő módon történt:

$$hasonlóság = 100 - \frac{100 * különböző\ sorok\ száma\ (diff)}{első\ fájl\ sorainak\ száma + második\ fájl\ sorainak\ száma}$$

A vizsgálat során a ’diff’ futtatása a –w opcióval kellett történjen, amely így figyelmen kívül hagyta azokat a sorokat, melyek mindössze a szóközök mennyiségében és/vagy pozíciójában különböznek.

A következő öt érték az öt vizsgált plágiumdetektáló szoftver által számított hasonlósági érték. A Marble kivételével a vizsgált eszközök kettő pontszámot adnak vissza. Az egyik pontszám azt mutatja, hogy az ’A’ fájl sorainak hány százaléka hasonlít a ’B’ fájl soraihoz, míg a másik éppen fordítva. Ezeknél az eszközöknél a két hasonlósági pontszám közül a nagyobb érték számít mérvadónak, ezáltal az összehasonlítás során használatosnak.

Az eredmények értelmezése

Nem áll rendelkezésünkre adat arról, hogy egy adott eszköz milyen függvényt használ pontszámainak számításához. Következésképpen az, hogy az X eszköz magasabb eredményt mutat egy verzióra az Y eszköznél nem jelenti azt, hogy az Y eszköz sokkal érzékenyebb az efféle támadásokra, mint az X eszköz.

Ráadásul, a plágiumdetektáló eszközök használatának egyik módja azok futtatása egy beadványhalmazon, majd a program által gyanúsnak jelölt párok manuális vizsgálata, kezdve a legmagasabb pontszámú pártól az egyre kisebb pontszámú párokon át egészen addig, amikor a jelöltek között már nem találunk több gyanús esetet. Az ilyen vizsgálatok során a konkrét eredmények nem fontosak. Ettől függetlenül néhány érdekes információt leszűrhető az első és második ábrából. Az eszközök refaktorizálási érzékenységének értékelése egy százas skálán történik. További elemzésre érdemesek lehetnek azok az eszközök, amelyek:

- nagyon alacsony pontszámot érnek el,
- a ’diff’ eszközzel meghatározott pontszámnál kisebbet érnek,
- a többi eszköztől szignifikánsan eltérnek az eredményei.

Az első és második ábra grafikonjaiból több érdekesség is megfigyelhető. Az egyes verziók, illetve kapcsolódó verziók csoportjainak megfigyelése együtt kell történjen! Továbbá minden egyes verzió és minden egyes eszköz esetén választani kell a két ábrán látható eredmények közül, hogy a lehető legjelentőségteljesebb pontszámot kapjuk. Például a QSortAlgorithm.java fájl hetes, nyolcas, kilences, tizenötös és tizenhatos verziójának minden pontszáma száz – ennek oka, hogy ezek a refaktorizálások nincsenek hatással magukra a fájlokra. Ellenben a QSortApplet.java fájlokban okoznak némi változást, ezért ezen verziók esetében az első ábra pontszámait kell mérvadónak tekinteni.

Kezdjük az általános észrevételekkel. Az öt eszköz közül négy elég nagy rendszerességgel ér el 100as pontszámot, sőt, alkalmanként még a diff is. Ugyanakkor a MOSS soha nem ér el 100as pontszámot. Mivel ez megtörténik azonos fájlokra is, ez egy lényeges tulajdonsága lehet a MOSS-nak, és emiatt a kilencvennyolc és kilencvenkilenc értékek száznak tekinthetőek.

- Először is mindkét ábrán látható az első verzió pontszámaiból, hogy az összes eszköz érzéketlen a kommentek és az elrendezés változásaira (szigorúan véve a MOSS nem teljesen érzéketlen).
- A kettes, hármas és négyes verzió sávjai a metódusok helyzetére tanúsított érzékenységet mutatják. Kizárólag a Marble teljesen érzéketlen a metódusok helyének megváltoztatására. A második ábrán megfigyelhető, hogy a JPlag, MOSS, Plaggie és a SIM jelentősen érzékeny a metódusok helyében bekövetkezett változtatásokra. Ugyanezen az ábrán látható, hogy az eszközök által adott hasonlósági pontszámok enyhén csökkennek, amikor az osztály attribútumok helyében történt változás.
- Egyik eszköz sem teljesen érzéketlen az osztály attribútumok helyében bekövetkezett változásokra (ötös és hatos verzió)
- Az első ábrán látható, hogy a hetes verziónál a Marble kivételével minden eszköz jelentősen alacsonyabb pontszámot ér el, mint a ’diff’. Ebben a verzióban a grafikus felhasználó felület kódja esett át refaktorizáláson.

- A nyolcas, kilences, tízes és tizenegyes verziókban végrehajtott változtatások teljesen hatástalannak bizonyultak mindegyik eszközzel szemben, mindkét fájl esetében (egy kis kivétellel a JPlag és MOSS esetében). Ezek a verziók a változók és osztályok átnevezését, valamint az importálások változtatását érintik.
- A tizenkettes verziókban elkövetett változtatások láthatólag kisebb zavart okoztak a Marble esetében, és nagyobbat a MOSS és a SIM esetében. A JPlag és a Plaggie érzéketlen a refaktorizálás ezen módjára.
- Majdnem ugyanez vonatkozik a tizenhármas verzióra is – ez az Eclipse kódtisztító funkciójának alkalmazása után létrejött verzió. Ugyanakkor itt már a Plaggie pontszámai is esnek.
- Mind a JPlag, mind a Marble láthatóan rosszul teljesített a tizennégyes verzión (Eclipse által generált hash kódok és equals funkciók a forráskódban): alacsonyabb pontszámot mutatnak, mint a ’diff’. Ugyanakkor a MOSS, a SIM, és meglepő módon – mivel nagyon hasonlít a JPlagra – a Plaggie is jól teljesített ezen a verzión.
- A tizenötös verzió esetében egyik eszköz esetében sem beszélhetünk érzéketlenségről. A Marble és a JPlag magasabb pontszámot ad, mint a ’diff’, a másik három viszont alacsonyabbat.
- A JPlag, a Plaggie és a SIM érzéketlenek a tizenhatos verzióban fellelhető refaktorizálásra, míg a Marble és a MOSS láthatóan nem. Ugyanakkor, mint az mindkét ábrán látható, a Marble mindkét fájl esetében a ’diff’ pontszámánál magasabb pontszámot ad.
- Tizenhetes verzió (get és set metódusok generálása): egyik eszköz sem teljesen érzéketlen az ilyen típusú refaktorizálásra, de a Marble és a JPlag esetében ez láthatóan sarkalatos pont lehet, mivel mindkettő alacsonyabb pontszámot ért el a ’diff’-nél a QSortAlgorithm.java esetében.

Első pillantásra levonhatónak tűnik az a következtetés, miszerint – bár vannak különbségek az osztályozásban – minden próbálkozást, mely a plágium elrejtésére irányult, valamennyi eszköz megtalált volna. Összességében az eszközök a legtöbb verzió esetében magas hasonlósági értéket adtak vissza. Illik megjegyezni, hogy az előző állítás érvényessége nagymértékben függ attól, hogy az egyes eszközök hogyan értékelik a plágiummentes eseteket. Megtörténhet ugyanis, hogy az eszközök a plágiummentes párokat is hasonlóan

értékeli, mint azt az elvégzett kísérletben is láthattuk. Ennek ellenőrzése egy másik kísérlet keretein belül történhet meg. Ugyanakkor, az eszközökön most elvégzett kísérletek által szerzett tapasztalatok alapján kijelenthető, hogy az eszközök többsége látszólag sokkal kisebb pontszámmal értékeli a nem hasonló párokat, mint az érzékenységi vizsgálatokban látott pontszámok.

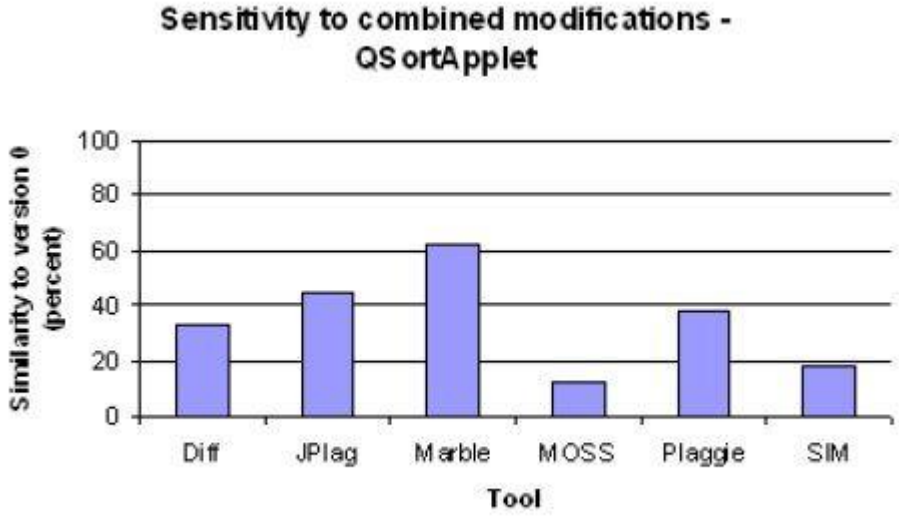
ii. Érzékenység refaktorizációk kombinációjára

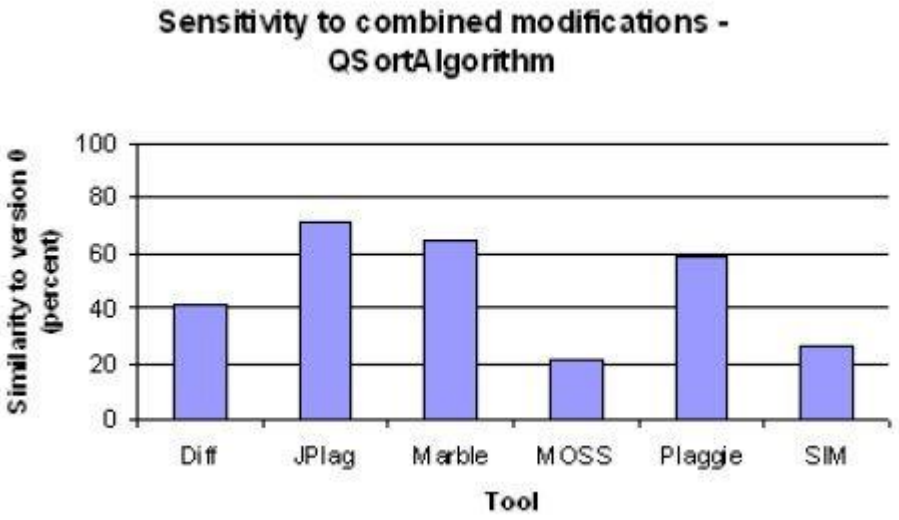
Beállítások és a vizsgálat eredményei

Az előző kísérletet egy új szintre emelve, számos hatékonynak bizonyult módszert kombinálva egy új forráskóddal történik a rendszerek támadása. Konkrétan a négyes, hatos, és a tizenegytől tizenhétig terjedő verziókban alkalmazott változtatások kombinálása történik egy új forráskód formájában. Ezek a módszerek bizonyultak a leghatékonyabbnak a legtöbb eszköz eseten. A hatékonyság mellett a tizenegyes-tizenhetes számú változtatások könnyedén véghez vihetőek, hiszen azok végrehajtása automatikusan történik az Eclipse által. A kérdés, melyre ez a kísérlet a választ keresi: ezáltal elkerülhető-e a plágium detektálása? Az eredmények a hármas számú ábrán láthatóak.

Az eredmények értelmezése

A hármas számú ábrán látható eredményeken látható, hogy mind a MOSS mind a SIM esetében a hasonlósági pontszámok mindkét fájl esetében nagyot csökkent. A Marble, a JPlag és a Plaggie esetében a hasonlósági pontszámok csökkenése kisebb méreteket ölt. Ahhoz, hogy a plágium ténye lelepleződjön, elég a beadvány egy fájljának szerepelnie a rangsor elején. Nagy biztonsággal állítható, hogy a Marble által 64 pontosra értékelt QSortAlgorithm.java az eredmények listázásakor meglehetősen elől lesz.





Harmadik ábra

V. Teljesítmény nagy programhalmazon

Az eszközök teljesítményének összehasonlításának másik módszere az eszközök első n darab kimenetének vizsgálata. A következő részben két ilyen kísérlet eredményeit vizsgáljuk.

a. Első kísérlet

Az első kísérletben az eredményeket n=10 esetén vizsgáljuk. Ez azt jelenti, hogy a különböző eszközökön történő futtatás során 10 legmagasabb pontszámot elért beadvány párok vizsgálata történik meg manuálisan. Ezáltal minimum 10 (feltételezzük, hogy van 10 eléggé hasonló pár, melyeket legalább egy eszköz gyanúsnak ítél), maximum 50 (mivel 5 különböző eszközzel végzünk vizsgálatot) pár programot kell megvizsgálni: vajon a programok természetes módon hasonlóak, vagy a hasonlóság megalapozottan plágiumból származik.

A nagy hasonlóságot mutató párok összesítésre kerülnek. Azok az eszközök tekinthetőek jónak, melyek a 10 legmagasabbra pontozott plágium gyanús esetek között a legtöbbet tartalmazzák ezekből a párokból – azok közül is inkább az igazi plágium eseteket. Például ha egy X refaktorizálási technika nem használatos a beadványok körében a plágium elrejtésére, akkor két, Y és Z eszköznek elméletileg ugyanolyan precíznek kellene lennie, ugyanakkor megtörténhet, hogy Y eszköz könnyedén észleli a plágium elrejtését, míg a Z reménytelen ilyen téren.

Amikor egy eszköz egy forráskód párt a hasonlósági lista elejére helyez, de a vizsgálat során a két forráskódról kiderül, hogy nem hasonlóak, az egyértelmű jel arra, hogy még van lehetőség a program további fejlesztésére, kiváltképpen ha az adott kombinációt egyéb eszközök nem listázzák a gyanús esetek között.

Beállítások:

Az összehasonlítás alapját az érzékenységi vizsgálathoz képest egy másik gyűjtemény jelenti. Az Utrechti Egyetemen meglehetősen nagy mennyiségben lelhetőek fel a Mandelbrot halmazokhoz kapcsolódó feladatra hallgatók által benyújtott megoldások, egészen a 2002-

2003-as tanévig visszamenően. A kollekcióban bizonyítottan található néhány plágium eset. A kísérlet során ezek a beadványok kerülnek vizsgálat alá, megfelelően könyvtárakba rendszerezve a hamis riasztások elkerülésének érdekében.

Az eredmények ismertetése

A negyedik táblázat mutatja a kísérlet eredményeit. Mind az öt eszköz esetében a tíz legmagasabb hasonlósági pontszámmal rendelkező beadvány pár megtalálása volt a cél. Tudni kell, hogy ez az összehasonlítás a kimeneti fájlok összehasonlíthatóvá konvertálása után készült. Ehhez a konvertáláshoz szükség van külső eszközre, különösképpen akkor, ha a kísérletet később egy másik beadványhalmazon megismétlésre kerülne. A konverziót végző eszközök bizonyítékot szolgáltat, hogy a különböző eszközök által számított pontszámok kezelése egyforma, valamint dokumentálja a konverzió módját. A konverzió manuális végrehajtása nem szolgál ilyesfajta bizonyítékkal.

Ezek után 28 különböző pár marad, melyek a negyedik táblázatban láthatóak, mindenféle rendezés nélkül felsorolva. A sorok számozása csak a könnyebb hivatkozás megteremtése miatt történt.

Ezen felül a kérdéses párokra vonatkozó eredmény nem csak jelölve, minősítve is lett. Hat különböző minősítés található a táblázatban:

- ’plágium’: a kérdéses pár megerősített plágium eset.
- a ’téves riasztás’ és ’hasonló’ minősítés között csak egy hajszálnyi különbség van: a hasonlónak címkézett párok tartalmaznak néhány fájlt, melyek természetükből kifolyólag hasonlóak, míg a téves riasztás esetében a hasonlóságról – és ezáltal plágiumról – szó sincs.
- az ’újrabeadott’ minősítés az adott program legálisan történt újra beadására vonatkozik: ugyanazon diák egy másik tanévben.
- 3 olyan pár található, melyek minősítése ’ugyanazon beadvány’: természetesen ezek azonos beadványok, nem plágiumesetek. Csak a SIM talált rá ezekre a párokra, ami viszont nem különösebben meglepő, hiszen a SIM egy klón kereső eszköz, ami indokolja az önmagával történő összehasonlítást.

- Végül pedig a ’kis fájl’ minősítés, mely azt jelzi, hogy a hasonlóság abból adódik, hogy a szóban forgó fájlok nagyon kisméretűek, ezekben az esetekben nincs plágium.

	JPlag	Marble	MOSS	Plaggie	SIM
1 plágium		86	46	60	30
2 plágium	elemzési hiba	82	49	elemzési hiba	
3 plágium	94	94	72	89	70
4 téves riasztás		43		94	4
5 téves riasztás		44		94	96
6 téves riasztás		44		94	
7 téves riasztás		44		94	
8 újrabeadott	95	92	60	92	61
9 hasonló	100	84	87	100	100
10 újrabeadott	100	100	84	100	
11 téves riasztás		44		94	
12 téves riasztás	84	56		65	
13 téves riasztás	84	52		69	
14 téves riasztás	85	57		69	
15 újrabeadott	100	100	73	100	100
16 téves riasztás		47		94	
17 plágium	elemzési hiba	92	71	elemzési hiba	73
18 téves riasztás		28			83
19 plágium	84	89	49	92	56
20 téves riasztás	86	54			
21 ugyanazon beadvány					100
22 hasonló	79	94		90	43
23 kis fájl		31			97
24 téves riasztás	96	89	80	97	75
25 ugyanazon beadvány					100
26 téves riasztás	83	61	63	79	63
27 hasonló	85	75		82	49
28 ugyanazon beadvány					100

Negyedik táblázat

A táblázat összesen 13 releváns párt tartalmaz, melyek magas hasonlóságot mutatnak és érdemes lenne őket továbbvizsgálni – ezek a ’plágium’, ’újrabeadott’ vagy ’hasonló’ minősítéssel ellátott párok. 4 irreleváns pár található – ’ugyanazon beadvány’ vagy ’kis fájl’ minősítéssel –, melyek nem tartoznak egyik top tízbe sem. A további 11 párnak – a téves riasztásoknak – nem igazán lenne helye egyik top tízben sem.

Az első, második és tizenhetedik sor kapcsolódnak egymáshoz három beadvány révén, amik legyenek jelölve A, B és C programként, ahol A és B új beadvány, míg C egy másik hallgató által az előző évben beadott megoldás. Az első sor az (A,C), a második az (A,B), a tizenhetedik a (B,C) párokat reprezentálja. Az elemzési hibákat a B jelű program elemezhetetlensége okozza.

Az oszlopokban találhatóak az egyes eszközökhöz tartozó eredmények, amennyiben a pár szerepel az eszköz top tízében, vagy megtalálható volt az eredmény az outputban.Mint látható, a Marble minden párhoz rendelt eredményt – azokon kívül, melyek azonos beadványként lettek megjelölve – míg a többi eszköz nem. Ennek oka, hogy a Marble a vizsgálat során minden összehasonlított pár hasonlóságát pontozza, míg a többi eszköz csak egy bizonyos küszöb felett ad vissza hasonlósági értéket.

Az eredmények értelmezése

Az eredmények alapján megfigyelhető, hogy a JPlag, a Marble és a MOSS által leggyanúsabbnak titulált programok tízes listái hasonlóak egymáshoz, addig a Plaggie és a SIM eléggé különböző eredményeket adott vissza a másik három eszközhöz képest. A továbbiakban a top tízek részletesebb elemzése következik.

A JPlag teljesítménye: A sorok számottevő részében a JPlaghoz nem tartozik pontszám, melynek az okát érdemes lenne megvizsgálni. Az öt plágium esetből a JPlag kettőt figyelmen kívül hagy elemzési hibák miatt, és egyet, ahol egyszerűen nincs pontszám. A másik két eset pontszáma elég magas.

A Marble teljesítménye: A Marble elég jól teljesít, amit az is mutat hogy minden plágium gyanús, újra beadott, vagy hasonló párhoz legalább 82es pontszámot ad (kivéve a huszonhatodik és huszonhetedik sor, de ezen sorokban is még mindig nagyobb pontszámok láthatóak, mint a hasonló beadványok, téves riasztások vagy kis fájlok esetén).

A MOSS teljesítménye: A MOSS kettő kivételével minden olyan párt megtalált, melyek további vizsgálatra érdemesek. Néhány esetben az ilyen párok pontszáma inkább alacsonynak nevezhető, de mint azt már korábban említettem, ez lehet, hogy a MOSS egyik tulajdonsága. Más szavakkal kifejezve: lehetséges, hogy a MOSS küszöb értéke egyszerűen alacsonyabb a többi eszköz küszöbénél.

A Plaggie teljesítménye: A Plaggie meglehetősen sok esetben magas pontszámot ad vissza olyan esetekre, melyek téves riasztásnak bizonyultak. Következésképpen számos – valóban vizsgálatot igénylő – eset kerül ezek mögé a hamis riasztások mögé a rangsorban. Ezen kívül a Plaggie elemzési hiba miatt figyelmen kívül hagy két darab plágium esetet.

A SIM teljesítménye: A SIM pontszámai meglehetősen kiszámíthatatlannak tűnnek, látható ez például a plágium esetek, az újra beadott és hasonló beadványoknál. A SIM ezekre adott pontszámai a harminc és száz közötti intervallumban mozognak – harminc a plágium eseteknél, száz a hasonló és újra beadott eseteknél. Ezen kívül az ötödik sorban található téves riasztási eset nagyon magas, kilencvenhatos pontszámot kapott, igaz, ez az egyetlen téves riasztási eset, melyet a SIM fontosnak tekintett annyira, hogy lepontozza azt. A SIM top tízében az első három pozícióban ugyanazon beadványok vannak (ez a táblázatban a huszonnyolcas, huszonötös és huszonegyes sorokban látható). A huszonegyes sor ugyanakkor a SIM top tízének kilencedik helyén is szerepel. Ezeken kívül a kilences sor az ötödik és hatodik, a huszonhármas sor pedig a hetedik pozícióban található. Összefoglalva azt mondhatjuk, hogy a SIM hajlamos magas pontszámot adni olyan hasonlósági eseteknek, melyeknek nyilvánvalóan nem kellene a tíz leggyanúsabb eset között szerepelnie. Ennek egyik – fő – oka az, hogy a SIM néha a egy beadványt önmagához is hasonlít, ami – mint már említettem – abból adódik, hogy a SIM egy klón detektáló eszköz. Egy másik ok pedig az, hogy a SIM nem biztosít lehetőséget a bizonyos méretnél kisebb fájlok vizsgálatból történő kihagyására.

A MOSS és a SIM által kihagyott összehasonlítások: A SIM outputjából hiányzik a tízes sorban található pár összehasonlítása, míg a MOSS outputjában a huszonkettes és huszonhetes sorban található párokhoz nem találunk hasonlósági értéket. Utóbbi két páros további vizsgálatot igényelt (bár később kiderült, hogy egyik sem plágium eset). Lehetséges, hogy ez egy hiba az eszközökben, esetleg egy hiba az eszközök kísérlet során történt használatában. Ahhoz, hogy megállapíthassuk melyik eset áll fent, további vizsgálatra lenne szükség a SIM, a MOSS működésében.

Elemzési hibák: Az egyik beadvány esetén bekövetkezett elemzési hibák ellehetetlenítették a második és tizenhetedik sorban szereplő párokban jelen lévő plágium jelenlétének kimutatását mind a JPlag, mind a Plaggie esetében. Ezt az elemzési hibát nem az eszközök, hanem egy beadvány maga okozta. Szerencsére mindkét eszköz készít egy naplófájlt, melyben a két eset is említésre került. Ugyanakkor az is lehetséges, hogy ez egy

szándékosan beadott, szintaktikailag hibás beadvány. A kisebb hibák – mint például a (szándékosan) elfejtett záró zárójel – általában automatikusan kijavításra kerülnek a beadványt vizsgáló személy által. Az efféle trükkök kivédésének érdekében dönthet úgy valaki, hogy csak fordítható programokat hajlandó elfogadni.

b. Második kísérlet

Ebben a kísérletben a Debreceni Egyetem Informatikai Karának hallgatói által készített programok halmaza képezi a kísérlet alapját.

A feladat

A hallgatók harmadik beadandó feladata a 2009/2010es tanév őszi szemeszterében a csillapított Gauss-Newton módszer implementálása volt. A programokat C vagy C++ nyelven kellett elkészíteniük, az implementálandó feladat matematikai hátterét pedig egy online elérhető pdf fájl formájában tölthették le. A feladatra 113 diák nyújtott be megoldást, összesen 999 darabot. A benyújtott megoldások közül minden hallgató esetében az utolsó benyújtott program kerül vizsgálat alá. A megoldások természetesen egy fájlból állnak, melyek között a legterjedelmesebb megoldás mérete 29161 bájt: 929 sorból áll és 27305 karaktert tartalmaz. A legkisebb méretű megoldás 4156 bájt tárhelyet foglal, és 172 sorban 3814 karaktert tartalmaz. Ez utóbbi adatokból is látható, hogy a diákok programozási stílusa mekkora eltéréseket mutat, valamint, hogy a feladat megoldására számtalan különböző mód adódott.

A vizsgálat körülményei

Mivel a programok megoldása C illetve C++ nyelven kellett történjen, ezért az eddig vizsgált plágiumdetektáló eszközök közül ebben a kísérletben csak három használható – köszönhetően annak, hogy csak ezek támogatják az említett nyelveket. A kísérlet során tehát a programok vizsgálata a JPlag, a MOSS és a SIM rendszerek segítségével történt. A SIM esetében a honlapról letölthető csomagolt fájlban a SIM különböző verzióit találjuk: minden egyes támogatott nyelv vizsgálatához külön futtatható fájl áll rendelkezésünkre – én

természetesen a C nyelv vizsgálatához használatos verziót használtam. A rendszerek konfigurációja a vizsgálat során a következő volt:

- A SIMmel történő tesztelés esetén a vizsgálat testreszabása a karakteres felületen különböző argumentumok segítségével történik. A kísérlet során viszonylag nagy mennyiségű programmal történt a tesztelés, ezért az eredményeket az alapkonfigurációtól eltérő módon summázva szerettem volna megkapni, a többi eszköz eredményeivel történő hasonlítás miatt százalékos formátumban és természetesen a későbbi használat miatt az outputot fájlba irányítva. Ezek elérése rendre a –n, –p és –o kapcsolók használatával állítható be. Ezen kívül alapvető fontosságú volt, hogy a fájlokat önmagukkal ne hasonlítsa össze a rendszer: ezt a –s kapcsoló használatával érhetjük el. A felsoroltakon kívül kényelmi szempontok miatt a –T kapcsoló használatával az eredmények tömörebb és egységesebb formába öntését is aktiváltam.
- A MOSS rendszer esetében a regisztráció után kapott PERL kóddal – illetve annak valamilyen formában futtathatóvá tételével – történik a feltöltés a MOSS szerverére, ahol a vizsgálat maga megtörténik. A kód tartalmazza a felhasználási és telepítési instrukciókat, valamint a szerzői jogokra vonatkozó figyelmeztetést is. A kód lényegi része pedig a szükséges változtatások esetére megfelelő módon kommentelve van. A MOSS esetén az egyetlen kapcsoló amire szükségem volt a vizsgálat során az a –l, melynek segítségével megadható a vizsgálandó fájlok nyelvezete.
- A JPlaggal történő vizsgálathoz az eszköz honlapján található Web Start kliens használatára, és egy JPlag accountra van szükség. A kliens elindítása és a bejelentkezés után hozhatunk létre egy új vizsgálatot, melyben a vizsgálandó fájlok megadását követően elérhetőek a beállítások. A JPlag beállításai közül a nyelvre, és az eredmények kimutatására vonatkozóakat állítottam át.

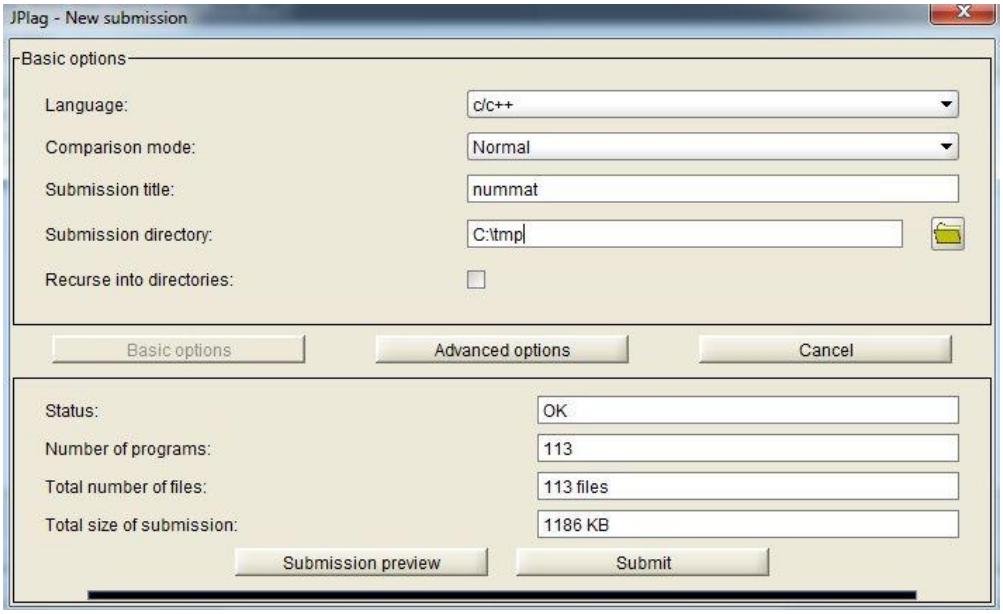
A vizsgálat menete

A vizsgálat megkezdése előtt a 999 darab beadvány közül kiválasztottam az egyes diákokhoz tartozó utolsó beadványokat, így megkapva a 113 vizsgálat alá vetendő programot, és azokat elkülönítettem egy könyvtárba.

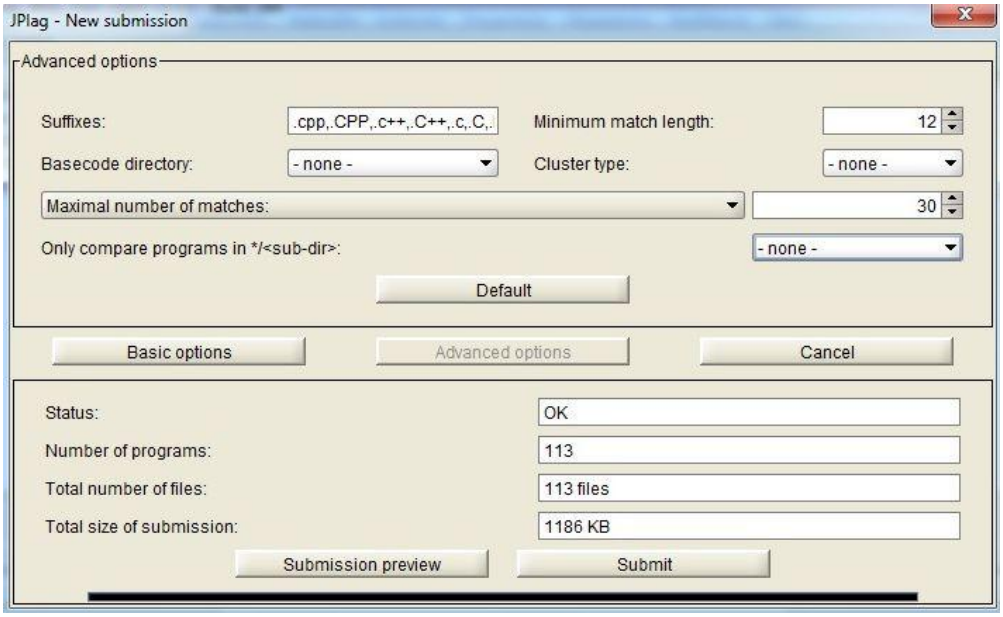
- Vizsgálat SIMmel: SIMmel a vizsgálat végrehajtásához mindössze a program parancssoron keresztül történő futtatására van szükség – természetesen a megfelelő paraméterek és a vizsgálandó fájlok argumentumként történő megadásával. A vizsgálat folyamata során – amennyiben semmilyen hibát nem vétettünk a program indításakor – semmiféle visszajelzést nem kapunk arról, hogy a vizsgálat éppen melyik fázisában tart. Ha a programot nem megfelelő paraméterekkel szeretnénk futtatni, arról a program hibaüzenet formájában tájékoztat minket, a hiba konkrét visszajelzésével. Ilyen hiba lehet a kapcsolók nem megfelelő használata: nem létezők kapcsolók használata (ekkor a program kilistázza a használható kapcsolókat és azok hatásait), vagy a kapcsolók nem megfelelő módon történő halmozása illetve alkalmazása (például aktiváljuk az argumentumok standard inputról történő megadását, mégis megadjuk a fájlok nevét a parancsban). Az azonos kapcsolók halmozása egyébként a program futására nincs hatással. A vizsgálat befejezése után – ha nem rendelkeztünk róla másképp – a program kiírja az eredményeket a képernyőre és automatikusan kilép.
- Vizsgálat a MOSSal: A MOSSal történő vizsgálat végrehajtásához SIMhez hasonló módon a program megfelelő argumentumokkal történő futtatására van szükség. Ellentétben viszont a SIMmel a standard outputon a vizsgálat folyamán is kapunk némi visszajelzést a folyamat állapotáról. Értesítést kapunk a fájlok vizsgálatáról, és azok által esetlegesen okozott hibáról. Ilyen hiba adódhat, ha az argumentumokban megadott, vizsgálandó fájlok nem léteznek, vagy nem olvashatóak. Ekkor természetesen a program hibával leáll, és kiírja nekünk a hiba típusát, és az érintett fájlra vonatkozó argumentumot. A program futásának következő fázisa a fájlok feltöltése a szerverre. Ha nem sikerült kapcsolatot létesíteni a szerverrel arról szintén hibaüzenetet kapunk és a program kilép. A fájlok feltöltése egyenként történik, és erről visszajelzés is egyenként jelenik meg

a képernyőn. Sikeres beadás után pedig – némi várakozást követően – visszakapjuk az eredményeink eléréséhez szükséges URL-t. Egyébként a MOSS a SIMhez hasonlóan nem érzékeny a kapcsolók szabályos módon történő halmozására.

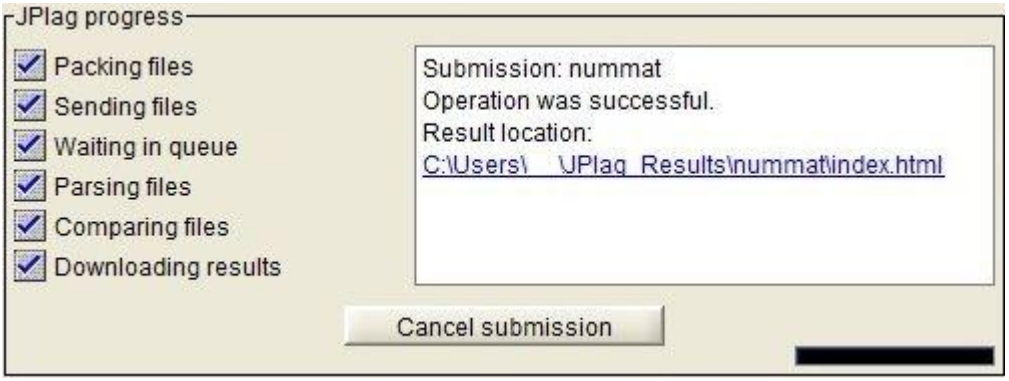
- Vizsgálat a JPlaggal: A JPlag használatához szükséges klienst letöltjük és bejelentkezünk. Ezután készítünk egy új beadványt. Az első képernyőn meg kell adjuk a vizsgálandó forráskódok programnyelvét, és az azokat tartalmazó könyvtár elérési útját. Megadhatjuk a vizsgálat módját (normál/revízió), valamint a vizsgálatnak nevet is adhatunk – amennyiben nem tesszük a program saját magának generál egyet. A forráskódokat tartalmazó könyvtár megadása után a JPlag megvizsgálja a vizsgálandó beadványokat, és egy statisztikát is rendelkezésünkre bocsát azok számáról, az összes vizsgált fájl darabszámáról és beadvány teljes méretéről. Lehetőségünk van haladó beállítások változtatására is. Itt változtathatjuk meg az elfogadott fájlkiterjesztéseket, a minimum fájlhosszúságot amit hasonlítunk, az alapkód könyvtárát – ha használunk báziskódot – , az eredmények csoportosításának típusát, az eredmény oldalon kiírt hasonlóságok maximális számát, vagy minimális hasonlósági küszöbét, és szükség esetén azt az alkönyvtárat, amelyet kizárólagosan vizsgálni szeretnénk. A beadás előtt lehetőségünk van a beadvány áttekintésére, melyben a JPlag kilistázza számunkra az általa felismert struktúrát a beadványban, valamint a vizsgálatra alkalmatlan fájlokat. Miután elindítjuk a vizsgálatot, egy grafikus ablakon keresztül tájékozódhatunk a folyamat állapotáról. Ez alapján a folyamatot hét részre bonthatjuk: a fájlok tömörítése, azok feltöltése a szerverre, várakozás a vizsgálat végrehajtására, a fájlok elemzése, azok összehasonlítása és végül az eredmények letöltése. Várakozás közben tájékoztatást kapunk arról, hanyadik helyen vagyunk a várakozók között. A vizsgálat végén tájékoztatást kapunk arról, sikeres volt-e a vizsgálat, és sikeres vizsgálat esetén az eredményeket tartalmazó HTML fájl elérési útjáról. A vizsgálatot bármikor megszakíthatjuk a folyamat során. Kissé zavaró tapasztalatom volt, hogy a vizsgálat végével nem tudtam bezárni a tájékoztató ablakot, ezáltal új vizsgálatot kezdeményezni sem – a program újbóli elindítása nélkül.



A JPlag alapbeállítások képernyője



A JPlag haladó beállítások képernyője



A JPlag folyamatkísérő képernyője

Az eredmények ismertetése és értelmezése

A három eszköz által elvégzett vizsgálatok eredményei alapján egyértelműen kijelenthető, hogy a beadványok között egy darab plágium pár van – ez megegyezik a beadványokat vizsgáló oktató által kapott eredménnyel. Ennek a fájlpárnak a hasonlósági mértékét a három rendszer a következőképpen pontozta: JPlag – 92.6%, SIM – 91%, MOSS – 80%. Bár nem ismerjük a másoló hallgató erőfeszítéseit, kijelenthető, hogy nem állt messze attól, hogy a plágium tényét a MOSS rendszere elől elrejtse. Ezen a páron kívül szignifikáns hasonlóságot egyik eszköz sem talált a beadványok között – egy közel 40%-os szakadékot tapasztalhatunk a következő leghasonlóbb pár pontszámáig. Feltűnő különbség, hogy míg a JPlag és a SIM hasonló pontszámokat ad az egyes párok hasonlóságára, míg a MOSS pontszámítása egy teljesen másik skálán mozog – az előző kísérlethez hasonlóan. Ehhez képest kissé meglepő, hogy míg a JPlag és a SIM a párok nagy részére 4% alatti hasonlóságot mutat, addig a MOSS eredményei között minden programpár hasonlósága legalább 4%.

90% - 100%	1	#
80% - 90%	0	.
70% - 80%	0	.
60% - 70%	0	.
50% - 60%	2	#
40% - 50%	4	#
30% - 40%	60	#
20% - 30%	286	####
10% - 20%	1183	##################
0% - 10%	4792	##

A Jplag eredmény oldalán található eloszlási táblázat

VI. Futási hatékonyság

A hatékonyság nagyon fontos tényező lehet a plágiumkereső eszközök esetében. Ezeket az eszközöket gyakran nagy mennyiségű beadványon alkalmazzák, melyek külön-külön több fájlt is tartalmazhatnak. Bár a kísérletek során nem készült konkrét feljegyzés a vizsgálatok időigényéről, elmondható hogy a kísérletek végrehajtói, nem tapasztaltak különösebb problémát az egyes rendszerek hatékonyságával. A futási idő a Java programok tesztelése során néhány perctől harminc percig terjedt, mely akadémiai körülmények között még elfogadható időtartamnak nevezhető. A C és C++ fájlok tesztelése során egyik rendszer futási ideje sem lépte túl az egy percet – a kommunikáció idejét beleszámítva sem. Ugyanakkor ezek az eszközök valószínűleg nem használhatóak egy olyan környezetben, ahol szorosabb időkorlátok vannak egy szignifikánsan komplikáltabb plágiumkeresési folyamat esetén. Érdemes megjegyezni azt is, hogy a plágium vizsgálat nem csak a programok futtatásának idejét emésztheti fel, hiszen ha valaki saját maga is leellenőrzi a gyanúsnak tűnő eseteket, az a program futási idejének többszörösére nyújtja azt az időt, ami a plágium keresés kezdetétől az eredmények előállásáig szükséges.

Összefoglalás

A szakdolgozatomban öt plágiumdetektáló eszközt vizsgáltam. Az eszközöket összehasonlítottam tíz funkció alapján, összehasonlításra került azok érzékenysége egy olyan programhalmazon, melyben mindegyik program szándékosan plagiarizálva volt, valamint egy olyan programhalmazon, mely a valós életben készült beadványokat tartalmazott, ezáltal összehasonlítva az eszközök teljesítményét. Az összehasonlítások eredményei jó rálátást adnak az egyes eszközök erős és gyenge pontjait illetően. Az eredményekből a következő következtetések vonhatóak le:

- Számos eszköz érzékeny számos kisebb változtatásra. A tizenkettes számú refaktorizálás példa egy ilyen átalakításra.
- Minden eszköz jól reagál a kombinálatlan refaktorizálások nagy részére, de a legtöbb rendszer rossz eredményt ér el refaktorizálási eljárások kombinálása esetén – néhány esetben rosszabbul, mint a ’diff’.
- Meglepő eredménye volt a top tíz összehasonlításnak, hogy a JPlag, a Marble és a MOSS eredményei meglehetősen hasonlóak, míg a Plaggie és a SIM eredményei meglehetősen nagy eltérést mutatnak a másik három rendszerhez képest.
- Fény derült néhány speciális esetre, melyek megoldásához további részletes vizsgálatra lenne szükség.

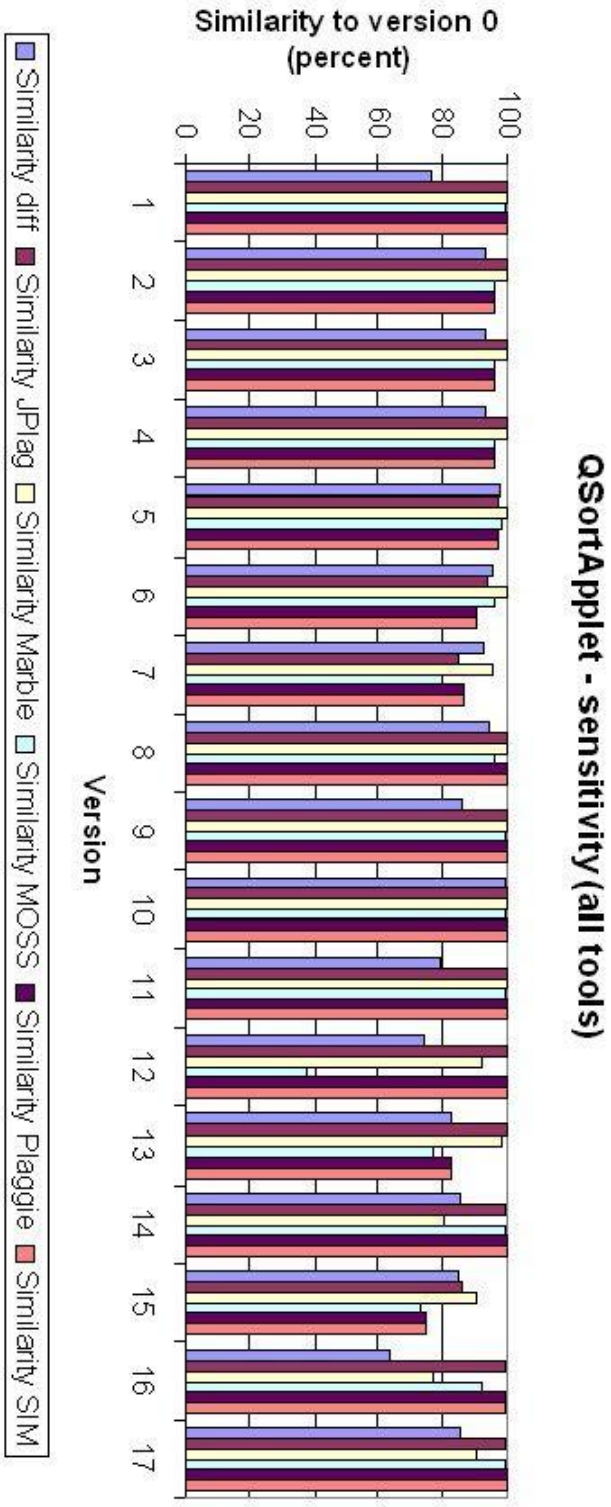
A személyes használat során szerzett tapasztalataim azt mutatják, hogy az egyes rendszerek használhatósága között hatalmas különbség van – természetesen a grafikus felületű JPlag előnyére. Karakteres felületen mind a beállítások, mind a vizsgálandó fájlok megadása körülményesebb, még huzamosabb használat után is. Ezen kívül a JPlag esetén több beállítás megváltoztatásával is konfigurálhatjuk a vizsgálatot, és az eredmény prezentációját, melynek áttekinthetősége és használhatósága is léptékekkel jobb a MOSS és a SIM rendszere által előállított eredményekéhez képest. Ezek az előnyök viszont nem ütnek vissza a JPlag sebességében. A eszközök érzékenység vizsgálata során legjobban teljesítő Marble hátránya, hogy csak a Java nyelvet támogatja – bár valószínűsíthető, hogy kiemelkedő eredményei is ennek a nyelvspecifikusságnak köszönhető.

A vizsgálatok alapján egyértelműen kijelenthető, hogy mind a plágiumot kereső, mind az azt elrejteni kívánó személyek sikerességéhez rengeteg erőfeszítésre van szükség. Kétségtelenül hasznos ezeknek a programoknak a használata az oktatók számára, de teljes mértékben nem zárható ki az emberi tényező a plágiumkeresés folyamatából, mindössze a szükséges erőfeszítés csökkenthető. Ugyanakkor felvetődik a kérdés – főleg a bárki számára elérhető –plágiumkereső szoftverek ellenkező irányban történő használatára, hiszen ha egy diák hozzáférést kap egy ilyen programhoz, felhasználhatja annak tesztelésére, hogy eléggé megváltoztatta-e a programját a plágium biztos elkerüléséhez.

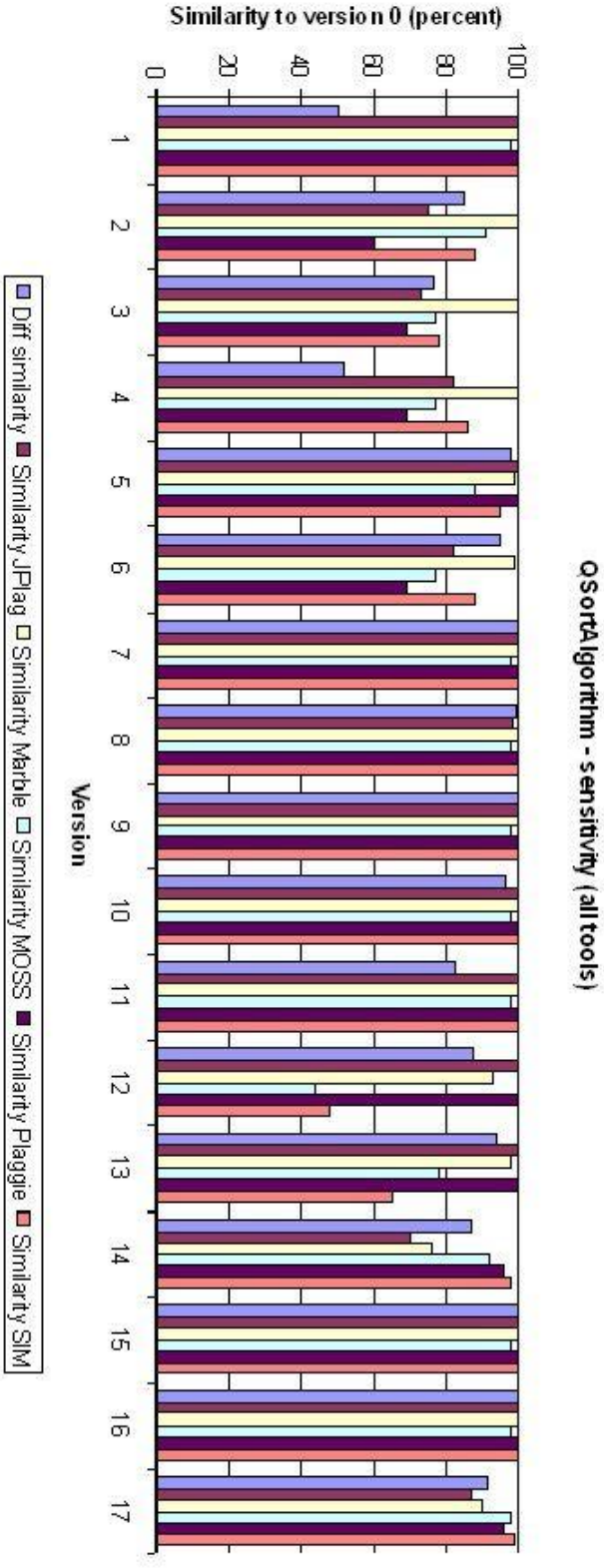
Irodalomjegyzék

- Aleksi Ahtiainen, Sami Surakka, Mikko Rahikainen: Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises.
- Thomas Lancaster, Fintan Culwin: A comparison of source code plagiarism detection engines.
- L. Prechelt, G. Malpohl, M. Philippsen: JPlag: Finding plagiarisms among a set of programs.
- Kevin W. Bowyer, Lawrence O. Hall: Experience Using ”MOSS” to Detect Cheating On Programming Assignments
- Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. Winnowing: Local algorithms for document fingerprinting.
- Mike Joy: Detecting Source-Code Plagiarism
- Paul Clough: Plagiarism in natural and programming languages: an overview of current tools and technologies
- http://theory.stanford.edu/~aiken/moss/
- http://www.cs.hut.fi/Software/Plaggie/
- http://wwwipd.ira.uka.de/jplag/
- http://dickgrune.com/Programs/similarity_tester/

Függelék



Első ábra



Második ábra

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindenkinek, akinek tanítása, munkája, segítsége hozzájárult ahhoz, hogy ez a dolgozat elkészülhessen.

Külön köszönet illeti Noszály Csabát a szakdolgozat elkészítése során témavezetőként nyújtott segítségéért.