Detecció automàtica de còpies en un conjunt de programes C

Ignasi Piñeiro Adell
Enginyeria en Informàtica

Joan Tous Nadal

12/06/2006

Als meus pares per la seva paciència

1. Resum

L'objectiu d'aquest projecte és analitzar les tècniques existents per a la detecció del plagi, i en concret aplicar-les a la detecció automàtica de còpies en un context acadèmic.

Actualment existeixen diferents aplicacions per a la detecció de plagis, durant el transcurs de la memòria s'estudiaran les solucions aportades per tal d'avaluar les eines i extreure'n les conclusions pertinents.

L'anàlisi previ ens servirà per introduir-nos en el món de la comparació de fitxers amb codi font, descriure els mecanismes existents i les seves particularitats.

Mitjançant el desenvolupament d'una aplicació per a detecció automàtica de còpies en un conjunt de programes C, aquest PFC[1] pretén estudiar un cas real de detecció de còpies en l'àmbit de la UOC[2] i arribar a una conclusió sobre la seva utilitat real.

Una vegada implementada l'eina es compararà amb les existents tant en qüestions de rendiment, com les avantatges que aporta a l'equip docent en termes d'utilització i representació de resultats.

El desenvolupament del projecte i estudi dels resultats obtinguts ens permet arribar a les següents conclusions:

- Existeix una gran preocupació en l'àmbit acadèmic de disposar d'una eina capaç de detectar plagis donat el nombre creixent de casos que s'estan produint en els darrers anys.
- La detecció automàtica de plagis és un camp complex, on intervenen factors de rendiment, representació de les dades, qualitat dels resultats obtinguts, etc.
- Un cop finalitzat el desenvolupament de l'eina es necessita temps per experimentar amb l'eina i aplicar diferents filosofies per tal de millorar la qualitat dels resultats obtinguts.

1.1. Paraules clau

Plagi, *tokens*, similitud, *Greedy String Tiling*, *tile*.

1.2. Àrea del projecte

El projecte s'inscriu dins de l'àrea E-learning: eines UOC.

[1] Projecte Fi de Carrera
[2] Universitat Oberta de Catalunya http://www.uoc.edu

2. Índex de continguts

3. Índex de figures

4. Introducció

4.1. Justificació del PFC i context en el qual es desenvolupa: punt de partida i aportació del PFC.

Molt sovint els estudiants d'enginyeria han de resoldre problemes amb llenguatges informàtics per avaluar els seus coneixements de programació així com del llenguatge en qüestió.

Aquestes pràctiques són realitzades per un gran nombre d'estudiants, cada any es dona el cas que estudiants copien les pràctiques, ja sigui utilitzant les de cursos anteriors, del mateix curs o realitzant la pràctica en comú quan aquestes tenen que ser individuals.

L'ús d'Internet ha afavorit el increment de casos de còpia ja que la distribució de les mateixes pràctiques és senzilla, a més s'ha incrementat l'obtenció de recursos electrònics.

Aquest PFC pretén realitzar un estudi a fons de les eines existents i desenvolupar una aplicació adaptada a les necessitats de la UOC per detectar casos de plagi entre els alumnes d'una assignatura, en concret les pràctiques desenvolupades en llenguatge C.

Per tant el punt de partida és l'estudi de les eines i mecanismes existents per a la detecció del plagi, i l'aportació del PFC és una eina que serveixi a l'equip docent de la UOC per detectar casos de plagi en la correcció de les seves pràctiques.

4.2 Objectius.

Els objectius del projecte són els següents:

- Introduir al lector en el món de la detecció del plagi, en concret en la comparació de fitxers de codi font.
- Instal·lar, avaluar i proporcionar un catàleg de les eines ja existents.
- Estudiar l'algorisme GST i mostrar el seu funcionament.
- Generar una eina automàtica de detecció de copies que pugui ser utilitzada per l'equip docent de la UOC.
- Finalment es comparà aquesta nova eina amb les existents i s'extrauran les conclusions pertinents.

4.3 Enfocament i mètode seguit.

El mètode seguit per aconseguir el objectiu principal del projecte, que és analitzar la detecció de plagis en un context acadèmic, ha estat utilitzar les eines existents i posteriorment desenvolupar una aplicació que reculli els requeriments de la UOC i apliqui les tècniques de la detecció de plagis. D'aquesta forma he anat analitzant les aportacions de cada eina, descobrint les particularitats de la comparació de fitxers i he aplicat les tècniques en un cas real.

Així podem dir que l'anàlisi de les eines i el desenvolupament de l'aplicació és el mètode utilitzat per a l'estudi de la detecció de plagis.

Per desenvolupar l'aplicació faig servir el model del Rational Unified Process amb notacions de l'UML[3], tal i com vaig aprendre en l'assignatura EPOO[4]. També han estat de vital importància els coneixements adquirits en les assignatures de Compiladors.

El cicle de vida del desenvolupament de l'aplicació és en **W** (*Bibl. 1*), com a simplificació del mètode en espiral ja que només realitzo dos cicles. És a dir que es farà un prototip a partir d'un disseny i un anàlisi.

Aquest prototip es valida i es verificarà amb l'usuari, en aquest cas els consultors del projecte. Una vegada aconseguida aquesta fita es realitzarà una revisió de l'anàlisi, disseny i s'implementarà el producte final.

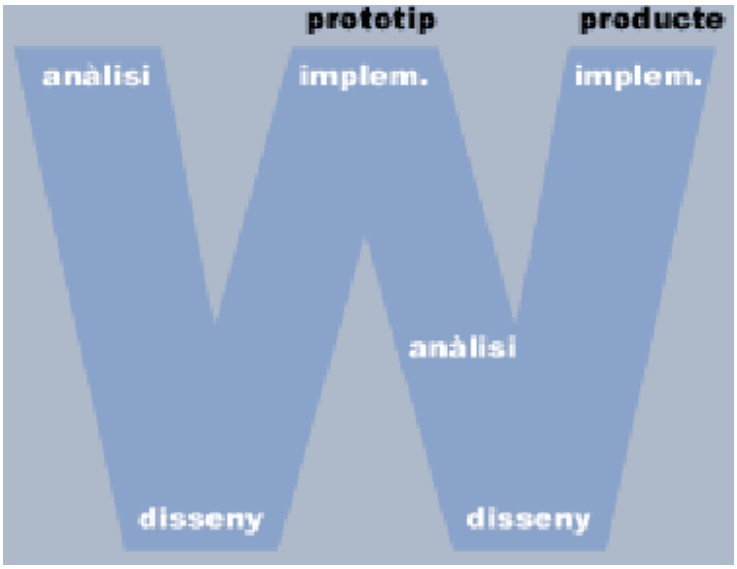


Figura 1 – Mètode de desenvolupament en W

[3] Unified Modeling Language http://www.uml.org
[4] Enginyeria del Programari Orientat a Objecte

4.4 Planificació del projecte.

A continuació es mostra la llista de tasques del projecte:

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
Gestió del Projecte	**74,13 días**	**mar 28/02/06**	**lun 12/06/06**	
Fites del Projecte	**74,13 días**	**mar 28/02/06**	**lun 12/06/06**	
T01 – Definició de l'abast del projecte	**7 días**	**mar 28/02/06**	**mié 08/03/06**	
Introducció	1 día	mar 28/02/06	mar 28/02/06	
Estructura del projecte i descomposició en tasques	2 días	mié 01/03/06	jue 02/03/06	9
Avaluació del esforç i duració de les tasques	2 días	vie 03/03/06	lun 06/03/06	10
Planificació temporal de les activitats	2 días	mar 07/03/06	mié 08/03/06	11
Entrega PAC1	0 horas	mié 08/03/06	mié 08/03/06	12
T02 – Detecció del plagi	**14 días**	**jue 09/03/06**	**mar 28/03/06**	**8**
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	jue 09/03/06	jue 09/03/06	13
Introducció	3 días	vie 10/03/06	mar 14/03/06	15
Cicle de vida per a la detecció de plagis	2 días	mié 15/03/06	jue 16/03/06	16
Factors crítics en la detecció de plagis	3 días	vie 17/03/06	mar 21/03/06	17
Tipus d'algoritmes per a la detecció del plagi	5 días	mié 22/03/06	mar 28/03/06	18
T03 – Avaluació de les eines existents	**27 días**	**jue 09/03/06**	**vie 14/04/06**	
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	jue 09/03/06	jue 09/03/06	12
Registre i instal•lació dels recursos necessaris	9 días	vie 10/03/06	mié 22/03/06	21
Funcionament de cada eina	1 sem	jue 23/03/06	mié 29/03/06	22
Jocs de proves	3 días	jue 30/03/06	lun 03/04/06	18;23
Anàlisi de les eines	9 días	mar 04/04/06	vie 14/04/06	24
T04 – Anàlisi de l'eina automàtica de plagis	**5 días**	**mar 04/04/06**	**lun 10/04/06**	
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	mar 04/04/06	mar 04/04/06	
Requeriments per al desenvolupament i integració de l'	2 días	mié 05/04/06	jue 06/04/06	27
Anàlisi funcional	2 días	vie 07/04/06	lun 10/04/06	28
T05 – Desenvolupament de l'eina	**21 días**	**mar 11/04/06**	**mar 09/05/06**	
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	mar 11/04/06	mar 11/04/06	
Creació d'una maqueta	9 días	mié 12/04/06	lun 24/04/06	31
Entrega PAC2	0 horas	lun 24/04/06	lun 24/04/06	32
Desenvolupament de l'eina	2 sem.	mié 26/04/06	mar 09/05/06	33
T06 – Proves	**14 días**	**mié 10/05/06**	**lun 29/05/06**	**30**
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	mié 10/05/06	mié 10/05/06	34
Jocs de proves	2 días	jue 11/05/06	vie 12/05/06	36
Estudi dels resultats obtinguts	3 días	lun 15/05/06	mié 17/05/06	37
Comparació de la nova eina amb les existents	3 días	jue 18/05/06	lun 22/05/06	38
Conclusions	4 días	mié 24/05/06	lun 29/05/06	39
Entrega PAC3	0 días	lun 29/05/06	lun 29/05/06	40
T07 – Manual d'instal•lació	**10 días**	**mar 30/05/06**	**lun 12/06/06**	**35**
Recol•lecta de material bibliogràfic relacionat (llibres, lr	1 día	mar 30/05/06	mar 30/05/06	41
Guia per a la correcte instal•lació	5 días	mié 31/05/06	mar 06/06/06	43
Exemple de funcionament	3 días	jue 08/06/06	lun 12/06/06	44
T08 – Preparació de la documentació final	**10 días**	**mar 30/05/06**	**lun 12/06/06**	**35**
Memòria final	3 días	mar 30/05/06	jue 01/06/06	41
Síntesi del projecte	7 días	vie 02/06/06	lun 12/06/06	47
Entrega Final	0 días	lun 12/06/06	lun 12/06/06	48

Figura 2 – Llista de tasques

4.5 Producte obtingut.

El producte obtingut és una eina de detecció de plagis, creada per al entorn de la UOC i es centra en els arxius de codi font del llenguatge C. L'entrada del programa seran les pràctiques dels estudiants i proporcionarà una sortida amb la informació sobre el tant per cent de similitud que hi ha entre les pràctiques.

L'usuari podrà navegar per el codi dels programes que s'estan comparant i es mostrarà ressaltada les parts que tenen en comú els programes, d'aquesta manera proporcionarem també una interfície perquè l'usuari pugui examinar millor les comparacions entre programes que han donat un grau elevat de copia i així doni el veredicte final.

També es mostrarà un gràfic per saber en quins percentatges han estat detectades les comparacions entre els programes.

4.6 Descripció dels capítols de la memòria.

Els capítols s'han organitzat de la següent manera:

- Detecció del plagi: es tracta de donar una visió general dels passos que s'han de seguir per aplicar els algorismes en la detecció del plagi i conèixer els factors importants de la mateixa.
- Avaluació de les eines existents: després recollir informació sobre les diferents eines que hi ha al mercat, s'escullen les més importants per avaluar-les.
- Anàlisi de l'eina automàtica de plagis: captura dels requeriments per al desenvolupament i integració de l'eina al entorn de la UOC.
- Desenvolupament: a partir del primer anàlisi es crearà un prototip. Un cop realitzat l'usuari el comprovarà i es farà un altre anàlisi per implementar les funcions desitjades i obtenir el producte final.
- Proves de l'eina automàtica de plagis: passar el joc de proves a la nova eina i comparar-la amb les eines existents.
- Instal·lació del software: elaboració d'un manual d'instal·lació de l'eina per a els usuaris que vulguin instal·lar-la en el seu propi computador i una guia d'utilització per l'usuari.
- Conclusions: recull final de les principals conclusions que s'han extret del projecte i valoració dels objectius aconseguits.

5. Detecció del plagi

5.1. Introducció.

Al llarg de la vida acadèmica els alumnes han de realitzar treballs i pràctiques de programació. Aquestes activitats estan orientades a que l'estudiant reflexioni per tal d'exercitar i desenvolupar els coneixements necessaris dels estudis en curs, així com definir el seu propi estil escrit, el vocabulari, gramàtica, etc.

Amb l'ús d'Internet s'ha incrementat l'accés a continguts digitals, de manera que es donen casos de copia total o parcial de treballs. En el cas que ens pertoca, el plagi es dona quan un alumne copia el codi font que ha d'entregar per resoldre una pràctica de programació.

Concretament, un exercici pràctic que han de realitzar en moltes assignatures els estudiants d'enginyeria informàtica, és fer un programa per a solucionar un enunciat plantejat pel professor. Els algoritmes a desenvolupar poder ser càlculs matemàtics, implementar estructures de dades, cerca en vectors, accedir a bases de dades, etc.

En funció de l'enunciat proporcionat pel professor l'alumne es troba amb més o menys llibertat a l'hora de desenvolupar el seu programa. Depenent de l'objectiu de l'assignatura i de la pràctica en concret ens trobem en casos en que el professor dona l'estructura del programa i els alumnes únicament han de implementar els cossos de les funcions, d'altres l'alumne té total llibertat i pot estructurar el programa com ell consideri millor.

Normalment el primer cas es dona en les assignatures de programació dels primers cursos i el segon cas en les dels últims cursos, ja que és tant important que l'estudiant conegui les bones metodologies a l'hora d'estructurar els programes com demostrar que ho sap fer des de zero.

Any rere any els professors es troben amb casos de plagi total o parcial en les solucions entregades per els estudiants.

A la pràctica si un professor no fa servir cap eina de detecció de plagi aquesta tasca li pot resultar molt pesada, el més probable és que detecti únicament els casos de copia més evidents. Per exemple quan un alumne entrega una pràctica en la qual s'ha deixat el nom d'un altre alumne o el corrector es dona compte que en una mateixa frase dos alumnes tenen la mateixa falta ortogràfica.

Aquests fets fan que el professor presti una major atenció i comenci a comparar amb més detall per tal d'esbrinar si es troba davant d'un cas de plagi.

Aquests casos de plagi podríem dir que són els més fàcils d'identificar, però existeixen d'altres que són més complicats i poden passar per alt. Per exemple

quan el infractor realitza el plagi amb cura i s'encarrega de modificar tots els comentaris, així com els noms de les variables, etc. En aquests casos una eina de detecció de plagis també ha de ser capaç de discernir si s'ha produït un cas de plagi, ja que l'estructura del programa serà la mateixa, i únicament cal que l'enunciat tingui uns mínims graus de llibertat perquè les solucions proposades per dos estudiants tinguin una estructura de codificació diferent.

Com anirem veient a continuació, les eines de detecció del plagi van més enllà de la comparació de dos fitxers font. Es poden detectar còpies encara que s'hagin realitzat modificacions en els noms de les variables, s'hagin afegit comentaris o modificacions en els noms de les funcions. Fan una comparació de l'estructura del programa i calculen el percentatge d'igualtat que tenen entre ells. En funció d'aquest percentatge es decidirà si s'ha produït un plagi.

El cas d'estudi es dura a terme en el plagi que es produeix en programes escrits en codi C.

5.2. Cicle de vida per a la detecció del plagi.

A continuació explicarem les transformacions que s'han d'aplicar a un programa per tal de poder aplicar els algoritmes de detecció del plagi.

Els passos a seguir es realitzen en tres fases:

1. Els programes que es volen comparar s'han de parcejar per convertir-los en una cadena de *tokens*.
2. Una vegada feta la transformació del programa en *tokens*, ja es pot passar a aplicar els algorismes de comparació per a la detecció del plagi per tal de determinar el grau de similitud que existeix entre ells. L'algorisme que faré servir és el *GST*[5] (*Bibl. 2*), el qual comentarem en detall en els propers capítols.
3. Després d'aplicar l'algorisme, s'ha de tenir cura de la representació que fem resultats. Un consultor pot tenir que corregir un nombre molt elevat de programes, set cents programes per exemple. Als ser un nombre molt elevat la representació dels resultats serà vital perquè l'usuari pugui discernir quins casos es consideren còpia i perquè. Així podrà fer la valoració final.

[5] Greedy String Tiling

Al següent diagrama és mostren els passos comentats anteriorment, així com les eines que s'utilitzen en cada part:

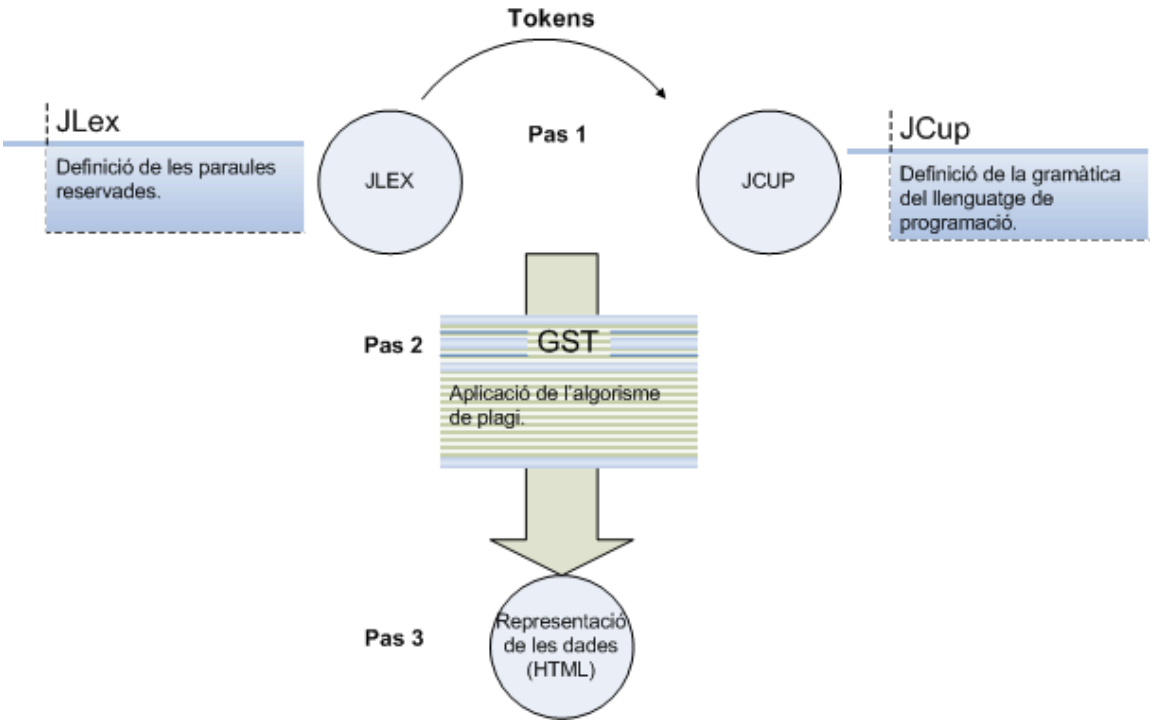


Figura 3 – Cicle de vida per a la detecció del plagi

5.3. Factors crítics en la detecció del plagi.

Per entendre els factors crítics en la detecció del plagis, es bo tenir una idea general de les tècniques que utilitzen els estudiants i que els impulsa a cometre aquest acte.

Les estadístiques diuen que el 80% els estudiants universitaris admeten que han plagiat part d'un treball o pràctica i el 36% han fet servir material copiat.

En el cas que ens pertoca que és el plagi de programes, s'ha de tenir en compte que en algunes assignatures al professor li és molt complicat plantejar una pràctica diferent a anys anteriors, de manera que els alumnes aconsegueixen les pràctiques passades i aquestes els serveixen de pauta per realitzar la que els correspon.

Per evitar que consultin les solucions d'any passats és important que el professor els transmeti la importància de realitzar la pràctica sense cap ajuda, ja que han de desenvolupar la capacitat de resoldre els problemes sense disposar d'una solució similar.

Un altre fet important que pot complicar la detecció del plagi, és que en les primeres assignatures que tenen els estudiants d'informàtica normalment quan es planteja una pràctica el professor dona part del codi. D'aquesta manera els alumnes només han d'omplir el contingut de les funcions, etc. Això és així perquè primer el estudiant veu com s'han de construir i estructurar els programes, també és més fàcil per el professor corregir la pràctica ja que les solucions estan més acotades.

Per tant hem de tenir present el següent, ens podem trobar en casos que les solucions estiguin molt acotades, en aquests casos una filosofia per detectar el plagi és que considerem copia aquelles solucions llunyanes a la solució del professor però molt pròximes entre si.

El primer factor que cal tenir en compte és com es durà a terme la transformació del programa a *tokens*. En un funció d'aquesta es podran detectar millor o pitjor, o segons una filosofia o altre els plagis entre programes. Es poden tenir en compte els comentaris que hi ha als programes o no, declaracions de variables, etc.

No hi ha unes regles definides per realitzar una traducció a *tokens*, més aviat aquesta es basa en la pràctica i l'experiència que es pot acumular a base d'utilitzar l'eina i adaptar-la a les necessitats d'una assignatura. Ja que com em comentat aquesta pot tenir uns condicions particulars.

Per exemple una traducció possible podria ser:

Codi C	**Tokens**
`#include <stdio.h>`	`Library`
`#include <stdlib.h>`	`Library`
`/* Inici main */`	
`int main()`	`BeginFunction`
`{`	
`  int n, b, p, r, c;`	`DefVar`
`  int i, exp;`	`DefVar`
`  scanf("%d", &n);`	`CallFunction`
`  while (n != 0) {`	`BeginWhile`
`    scanf("%d", &b);`	`CallFunction`
`    p = 0;`	`Assign`
`    r = 0;`	`Assign`
`    while (n !=0) {`	`BeginWhile`
`      c = n % b;`	`Assign Mod`
`      n = n / b;`	`Assign Div`
`      exp = 1;`	`Assign`
`      for (i = 1 ; i <= p; i++) {`	`Assign BeginFor`
`        exp = 10 * exp;`	`Assign Mul`
`      }`	`EndFor`
`      r = r + c * exp;`	`Assign Mul Add`

```
                p = p + 1;                          Assign Add
            }                                       EndWhile

            printf ("%d ", r);                      CallFunction
            scanf("%d", &n);                        CallFunction
        }                                           EndWhile

        return 0;                                   Return
}                                                   EndFunction
```

Amb la transformació present quan apliquem els mètodes de plagi no tindrem en compte els noms de les variables, els noms de les funcions, les operacions que es realitzen a dintre del while, etc. De manera que estarem comparant l'estructura dels programes. Com es pot observar en aquesta traducció no es tenen en compte els comentaris que hi ha al programa.

Aquesta política de traducció en alguns casos pot suposar una avantatge o anar en contra nostre, per exemple si la persona que ha realitzat la copia únicament s'ha limitat a canviar el nom de les variables i funcions, doncs detectarem la copia ja que l'estructura del programa seguirà sent la mateixa, de manera que una vegada aplicat el procés ens retornarà un percentatge molt alt d'igualtat.

El consultor en veure el resultat podrà comparar els dos codis, i segur que trobarà algun indici de copia en funció dels graus de llibertat que hagi donat quan va plantejar l'enunciat per realitzar la pràctica. Per exemple si troba que tots dos estudiants tenen els mateixos errors en els mateixos punts del algoritme, encara que el nom de les variables sigui diferent, podrà detectar que es troba davant d'un cas de còpia, i preguntant als alumnes segur que acaba traient l'aigua en clar i el infractor sortirà a la llum.

Aconseguir una millor traducció per als nostres propòsits només ens serà possible en base a l'acumulació d'experiència, ja que sabrem millor què volem detectar i com elaborem els enunciats per saber quines parts tenen limitades els estudiants.

En casos en que la solució és molt tancada, ja que el problema a resoldre es l'aplicació d'un algoritme conegut i el consultor ja dona part de l'estructura del programa, els casos de copia es poden trobar en els comentaris, per exemple en faltes que tinguin els alumnes, etc. es tracta de buscar casos molt similars i veure quins detalls poc habituals hi ha per determinar que s'ha produït una copia.

L'altre punt crític en la detecció del plagi es troba un paràmetre de l'algorisme *GST*, a continuació explicarem en detall el seu funcionament. En resum es tracta de determinar quina serà la mínima coincidència de *tokens* que ens tenim que trobar per considerar que és la mateixa en tots dos documents. És a dir, en funció d'aquest paràmetre per exemple si val 4, doncs per considerar que s'ha trobat un part igual de codi tindran que produir-se 4 igualtats seguides.

5.4. Tipus d'algorismes per a la detecció del plagi.

5.4.1. Greedy String Tiling:

L'algoritme que s'utilitza essencialment per comparar dos cadenes de tokens és el *Greedy String Tiling*. Quan compares dos cadenes A i B, l'objectiu del algoritme és trobar les sub cadenes més grans coincidents, aquestes sub cadenes coincidents s'anomenen *tiles* i han de complir les següents propietats:

- Cada token pot pertànyer com a molt a un tile. Una vegada utilitzat aquest es marca per evitar reutilitzar-lo en futures comparacions.
- Els tiles han de ser el més gran possibles, és bo trobar zones de coincidència com més grans millor.
- Els tiles han de tenir com a mínim la mida *Minimum Match Length*, que es defineix en el algoritme. En funció del valor d'aquesta variable els resultats de similitud varien.

A continuació mostrem el pseudocodi del algoritme (*Bibl. 2*):

```
GreedyStringTiling(Tokens A, Tokens B) {
      j = 0;
      minimumMatchLength = 4;
      maxMatch;
      tiles   = {};
      matches = {};
      do {
            maxMatch = minimumMatchLength;
            tiles = {};
            forAll unmarked Tokens A(i) in A {
                  forAll unmarked Tokens B(z) in B {
                        j = 0;
                        while( (A(i+j) == B(z+j)) && unmarked(A(i+j)) &&
                        unmarked(B(z+j) ) { j++; }
                        if(j == maxMatch) {
                              tiles = tiles U subToken(i, z, j);
                        }
                        else {
                              if(j > maxMatch) {
                                    tiles = {};
                                    tiles = tiles U subToken(i, z, j);
                                    maxMatch = j;
                              }
                        }
                  }
            }

            forAll tiles(i, z, j) in tiles {
                  mark(A, i, j);
                  mark(B, z, j);
                  matches = matches U subToken(i, z, j);
            }
      } while(maxMatch > minimumMatchLength);
      return;
}
```

Explicació del algorisme:

L'objectiu de l'algorisme consisteix en trobar les parts més grans coincidents en les dues cadenes de *tokens*, aquest ens assegura que els *matches* seran de tamany mínim *Minimum Match Length*.

La primera part del algorisme, el doble for, intenta trobar els tiles més grans possibles no marcats.

La segona part, el for, marca els tokens identificats en el matching actual.

El bucle principal, do while, assegura que es troben tots els tiles de mida Minimum Match Length.

En quant al cost computacional del algorisme, la part més costosa és la cerca de coincidències entre cadenes de tokens, per optimitzar aquest procés és important que la comparació de tokens no sigui entre Strings, sinó que cada token definit se li assigna un número enter únic, d'aquesta forma a l'hora de comparar tokens estarem comparant enters en comptes de Strings. Aquesta millora és la que proporciona l'algorisme *Karp-Rabin* (*Bibl. 3*).

Exemple d'execució:

A continuació mostrarem un exemple d'execució del algorisme. Aquest s'aplica als dos programes que mostrem a continuació amb la seva corresponent traducció a Tokens.

Programa 1	**Tokens**
`/*Programa canviBase*/`	
`#include <stdio.h>`	`Library`
`#include <stdlib.h>`	`Library`
`int main()`	`BeginFunction`
`{`	
`  int n, b, p, r, c;`	`DefVar`
`  int i, exp;`	`DefVar`
`  scanf("%d", &n);`	`CallFunction`
`  while (n != 0) {`	`BeginWhile`
`  scanf("%d", &b);`	`CallFunction`
`  p = 0;`	`Assign`
`  r = 0;`	`Assign`
`  while (n !=0) {`	`BeginWhile`
`  c = n % b;`	`Assign Mod`
`  n = n / b;`	`Assign Div`
`  exp = 1;`	`Assign`
`  for (i = 1 ; i <= p; i++) {`	`Assign BeginFor`
`  exp = 10 * exp;`	`Assign Mul`

```
                }                                       EndFor
                r = r + c * exp;                        Assign Mul Add
                p = p + 1;                              Assign Add
          }                                             EndWhile

          printf ("%d ", r);                            CallFunction
          scanf("%d", &n);                              CallFunction
     }                                                  EndWhile
     return 0;                                          Return
}                                                       EndFunction
```

Programa 2 **Tokens**

```
/* Canvi de base */

#include <stdio.h>                                      Library
#include <stdlib.h>                                     Library

int main()                                              BeginFunction
{
     int n,b,p,r,c,i,exp;                               DefVar

     scanf("%d" ,&n);                                   CallFunction
     while(n!=0)                                        BeginWhile
     {
          scanf("%d" ,&b);                              CallFunction
          p=0;                                          Assign
          r=0;                                          Assign
          while(n!=0)                                   BeginWhile
          {
               c=n%b;                                   Assign Mod
               n=n/b;                                   Assign Div
               exp=1;                                   Assign
               for(i=1;i<=p;i++)                        Assign BeginFor
               {
                    exp=10*exp;                         Assign Mul
               }                                        EndFor
               r=r+c*exp;                               Assign Mul Add
               p=p+1;                                   Assign Add
          }                                             EndWhile
          printf("%d ", r);                             CallFunction
          n=scanf("%d ",&n);                            Assign CallFunction
     }                                                  EndWhile

     return 0;                                          Return
}                                                       EndFunction
```

Tots dos programes implementen l'algorisme de canvi de base, cada un realitzat per un alumne diferent.

Una vegada tenim la traducció de cada programa a Tokens, el següent pas és aplicar l'algorisme *Greedy String Tiling* per obtenir les part coincidents del programa.

Inici:

A = { Library, Library, BeginFunction, DefVar, DefVar, CallFunction, BeginWhile, CallFunction, Assign, Assign, BeginWhile, Assign, Mod, Assign, Div, Assign, Assign, BeginFor, Assign, Mul, EndFor, Assign, Mul, Add, Assign, Add, EndWhile, CallFunction, CallFunction, EndWhile, Return, EndFunction }

B = { Library, Library, BeginFunction, DefVar, CallFunction, BeginWhile, CallFunction, Assign, Assign, BeginWhile, Assign, Mod, Assign, Div, Assign, Assign, BeginFor, Assign, Mul, EndFor, Assign, Mul, Add, Assign, Add, EndWhile, CallFunction, Assign, CallFunction, EndWhile, Return, EndFunction }

1 do:

tiles = { }
maxMatch = 4

Després del doble for:

tiles = { DefVar CallFunction BeginWhile CallFunction Assign Assign BeginWhile Assign Mod Assign Div Assign Assign BeginFor Assign Mul EndFor Assign Mul Add Assign Add EndWhile CallFunction }

maxMatch = 24

Després del for per marcar els tokens:

tile = { DefVar CallFunction BeginWhile CallFunction Assign Assign BeginWhile Assign Mod Assign Div Assign Assign BeginFor Assign Mul EndFor Assign Mul Add Assign Add EndWhile CallFunction }

A = { Library, Library, BeginFunction, DefVar, **DefVar**, **CallFunction**, **BeginWhile**, **CallFunction**, **Assign**, **Assign**, **BeginWhile**, **Assign**, **Mod**, **Assign**, **Div**, **Assign**, **Assign**, **BeginFor**, **Assign**, **Mul**, **EndFor**, **Assign**, **Mul**, **Add**, **Assign**, **Add**, **EndWhile**, **CallFunction**, CallFunction, EndWhile, Return, EndFunction }

B = { Library, Library, BeginFunction, **DefVar**, **CallFunction**, **BeginWhile**, **CallFunction**, **Assign**, **Assign**, **BeginWhile**, **Assign**, **Mod**, **Assign**, **Div**, **Assign**, **Assign**, **BeginFor**, **Assign**, **Mul**, **EndFor**, **Assign**, **Mul**, **Add**, **Assign**, **Add**, **EndWhile**, **CallFunction**, Assign, CallFunction, EndWhile, Return, EndFunction }

Matches = { DefVar CallFunction BeginWhile CallFunction Assign Assign BeginWhile Assign Mod Assign Div Assign Assign BeginFor Assign Mul EndFor Assign Mul Add Assign Add EndWhile CallFunction }

while: compleix la condició.

2 do:

tiles = { }
maxMatch = 4

Després del doble for:

tiles = { CallFunction EndWhile Return EndFunction }

maxMatch = 4

Després del for per marcar els tokens:

tile = { CallFunction EndWhile Return EndFunction }

A = { Library, Library, BeginFunction, DefVar, **DefVar**, **CallFunction**, **BeginWhile**, **CallFunction**, **Assign**, **Assign**, **BeginWhile**, **Assign**, **Mod**, **Assign**, **Div**, **Assign**, **Assign**, **BeginFor**, **Assign**, **Mul**, **EndFor**, **Assign**, **Mul**, **Add**, **Assign**, **Add**, **EndWhile**, **CallFunction**, **CallFunction**, **EndWhile**, **Return**, **EndFunction** }

B = { Library, Library, BeginFunction, **DefVar**, **CallFunction**, **BeginWhile**, **CallFunction**, **Assign**, **Assign**, **BeginWhile**, **Assign**, **Mod**, **Assign**, **Div**, **Assign**, **Assign**, **BeginFor**, **Assign**, **Mul**, **EndFor**, **Assign**, **Mul**, **Add**, **Assign**, **Add**, **EndWhile**, **CallFunction**, Assign, **CallFunction**, **EndWhile**, **Return**, **EndFunction** }

Matches = { DefVar CallFunction BeginWhile CallFunction Assign Assign BeginWhile Assign Mod Assign Div Assign Assign BeginFor Assign Mul EndFor Assign Mul Add Assign Add EndWhile CallFunction**,** CallFunction EndWhile Return EndFunction }

while: no compleix la condició.

Després de l'execució, ens ha trobat les següents parts coincidents:

Programa 1

```
Library
Library
BeginFunction
DefVar
DefVar
CallFunction
BeginWhile
CallFunction
Assign
Assign
BeginWhile
Assign
Mod
Assign
Div
Assign
Assign
BeginFor
Assign
Mul
EndFor
Assign
Mul
Add
```

Programa 2

```
Library
Library
BeginFunction
DefVar
CallFunction
BeginWhile
CallFunction
Assign
Assign
BeginWhile
Assign
Mod
Assign
Div
Assign
Assign
BeginFor
Assign
Mul
EndFor
Assign
Mul
Add
Assign
```

```
Assign
Add
EndWhile
CallFunction
CallFunction
EndWhile
Return
EndFunction
```

```
Add
EndWhile
CallFunction
Assign
CallFunction
EndWhile
Return
EndFunction
```

Com podem observar, al definir *Minimum Match Length* amb valor quatre la primera part coincident no ha entrat a formar part dels matches, perquè entres tindria que tenir com a valor mínim tres.

Un cop hem trobat els matches coincidents en els *tokens*, mostrem la seva representació en el codi dels programes:

Programa 1

```c
/*Programa canviBase*/

#include <stdio.h>
#include <stdlib.h>

int main()
{
  int n, b, p, r, c;
  int i, exp;

  scanf("%d", &n);

  while (n != 0) {
    scanf("%d", &b);
    p = 0;
    r = 0;
    while (n !=0) {
      c = n % b;
      n = n / b;
      exp = 1;
      for (i = 1 ; i <= p; i++) {
        exp = 10 * exp;
      }
      r = r + c * exp;
      p = p + 1;
    }

    printf ("%d ", r);
    scanf("%d", &n);
  }
  return 0;
}
```

Programa 2

```c
/* Canvi de base */

#include <stdio.h>
#include <stdlib.h>

int main()
{
  int n,b,p,r,c,i,exp;

  scanf("%d" ,&n);
  while(n!=0)
  {
    scanf("%d" ,&b);
    p=0;
    r=0;
    while(n!=0)
    {
      c=n%b;
      n=n/b;
      exp=1;
      for(i=1;i<=p;i++)
      {
        exp=10*exp;
      }
      r=r+c*exp;
      p=p+1;
    }
    printf("%d ", r);
    n=scanf("%d ",&n);
  }

  return 0;
}
```

Després de trobar els matches, ens interessa trobar el grau de similitud (*Bibl. 4*) que hi ha entre ells, d'aquesta manera sabrem el tant per cent d'igualtat que tenen. Aquesta dada és molt interessant quan l'usuari compara un nombre elevat de

programes. Ja que sabrà amb facilitat en quines comparacions s'ha considerat que es produeix un grau de còpia més elevat.

A la pràctica serà la mesura que aportarà més informació a l'usuari per saber si s'ha produït un cas de còpia o no, o per mirar en més detall dues pràctiques que tenen un percentatge d'igualtat més elevat, així com per saber el percentatge d'igualtat que hi ha amb la solució del professor i les solucions de dos estudiants.

Hi ha diverses fórmules per obtenir el grau de similitud, a continuació explicarem les principals comentant avantatges i desavantatges que presenten.

Donat un programa X i un programa Y és defineixen les següents funcions simètriques:

1) **similitud1 (X,Y)** = tokens en comú / mitjana(mida X, mida Y)
2) **similitud1 (X,Y)** = tokens en comú / mínim (mida X, mida Y)
3) **similitud1 (X,Y)** = tokens en comú / màxim (mida X, mida Y)

S'observa que les tres funcions tenen el mateix numerador, en canvi el denominador ve donat per una funció que com arguments té la mida dels dos programes. D'aquesta manera s'aconsegueix que la funció de similitud sigui simètrica, és a dir que doni el mateix valor quan es compara el programa X amb el Y que quan la comparació és fa amb el programa Y i el X.

Un exemple de funció no simètrica seria per exemple si el denominador fos la longitud del programa que estem comparant.

1) **similitud (X, Y)** = tokens en comú / mida Y
2) **similitud (Y, X)** = tokens en comú / mida X

Com que les longituds dels programes que comparem no tenen perquè ser iguals ens trobaríem amb similituds diferents. Com hem comentat anteriorment aquest cas no ens interessa així que descartem les dos funcions anteriors.

A partir d'aquí és poden definir altres funcions de similitud com per exemple donar més pes als matches més grans, ja que no es el mateix trobar cinc matches de 10 tokens que trobar-ne només un de 50 tokens. En el segon cas el plagi és més probable.

Una manera de mesurar-ho seria utilitzant la següent funció:

1) **similitud2 (X, Y)** = sqtr(midaMatch1^2 + ... + midaMatchN^2) / mínim(mida X, mida Y)

En aquest cas si trobem 5 matches de 10 tokens en un fitxer de 100 tokens, el grau de similitud seria:

similitud2 (X,Y) = sqtr(5 * 10^2) / 100 = 22.3%

En canvi si trobem un match de 50 tokens, el grau de similitud seria:

similitud2 (X,Y) = sqtr(50^2) / 100 = 50%

Aquesta darrera mesura potser penalitza la similitud de manera excessiva, donat que si ens trobéssim dos matches, un de 70 tokens i un altre de 30 tokens tindríem:

similitud2 (X,Y) = sqtr(30^2 + 70^2) / 100 =76.1%

Segurament voldríem un percentatge de similitud més elevat, donat que els fitxers comparteixen 100% de tokens. Per això és podria definir utilitzant els dos criteris previs:

1) **similitud3 (X, Y)** = mitjana(similitud1, similitud2)
2) **similitud3 (X, Y)** = màxim (similitud1, similitud2)

Per decidir quina funció utilitzar únicament ens podrem basar en l'experiència per discernir quina funció mesura en funció dels nostres criteris el percentatge de similitud.

En concret definim les següents fórmules per saber a quina fem referència:

1) **similitudMitj (X,Y)** = tokens en comú / mitjana(mida X, mida Y)
2) **similitudMin (X,Y)** = tokens en comú / mínim (mida X, mida Y)
3) **similitudMax (X,Y)** = tokens en comú / màxim (mida X, mida Y)
4) **similitudMatch (X,Y)** = sqtr(midaMatch1^2 + ... + midaMatchN^2) / mínim(mida X, mida Y)
5) **similitudMitjMatch (X,Y)** = mitjana(similitudMitj, similitudMatch)
6) **similitudMaxMatch (X,Y)** = mitjana(similitudMax, similitudMatch)

Tornant a l'exemple anterior, ara mostrem el grau de similitud que tenen els dos programes:

Nombre total de tokens del programa 1: 32 tokens.
Nombre total de tokens del programa 2: 32 tokens.
Nombre de tokens en comú: 28 tokens.

similitudMitj (Programa1, Programa2) = 28 / mitjana(32, 32) = 87.5%
similitudMin (Programa1, Programa2) = 28 / mínim(32, 32) = 87.5%
similitudMax (Programa1, Programa2) = 28 / màxim(32, 32) = 87.5%
similitudMatch (Programa1, Programa2) = sqtr(24^2 + 4^2) / mínim(32,32) = 76.03%
similitudMitjMatch (Programa1, Programa2) = mitjana(87.5%, 76.03%) = 81.76%
similitudMaxMatch (Programa1, Programa2) = mitjana(87.5%,76.03%) = 81.76%

Una vegada realitzat el càlcul del percentatge de similitud ja hem finalitzat l'aplicació del algorisme. Podem mostrar a l'usuari una dada significativa, i aquest si vol entrar a navegar per el codi per veure quines parts s'han trobat iguales i donar la última valoració.

5.4.2. **WINNOWING**:

Aquest algoritme és similar a *Greedy String Tiling*. La diferencia radica en la forma de calcular els *hash*.

L'algoritme funciona generant *fingerprints* del document, que no és més que una selecció d'alguns hash de diferents fragments del document. A partir de determinats paràmetres que dona l'usuari, es construir fragments del document per generar un seguit de hash d'aquests fragments i d'entre ells triar segons unes regles que configuraran el *fingerprint* del document sencer.

Els paràmetres inicials són:

- t (*guarantee threshold*), indica la longitud mínima de la cadena plagiada.
- k (*noise threshold*), indica la longitud mínima que han de tenir les coincidències.

A partir d'aquests paràmetres es calcula el paràmetre w, que correspon a la mida de la finestra, aquest càlcul és fa amb la següent fórmula:

$$W = t - k + 1$$

Aquestes finestres defineixen de quina manera s'agrupen els hash.
A continuació mostrem el pseucodi de l'algorisme (*Bibl. 5*):

```
void winnow(int w /*window size*/)
{
      // circular buffer implementing window of size w
      hash_t h[w];

      for (int i=0; i<w; ++i) {
            h[i] = INT_MAX;
      }

      int r = 0; // window right end
      int min = 0; // index of minimum hash
      // At the end of each iteration, min holds the
      // position of the rightmost minimal hash in the
      // current window. record(x) is called only the
      // first time an instance of x is selected as the
      // rightmost minimal hash of a window.

      while (true) {
            r = (r + 1) % w; // shift the window by one
            h[r] = next_hash(); // and add one new hash

            if (min == r) {
            // The previous minimum is no longer in this
            // window. Scan h leftward starting from r
            // for the rightmost minimal hash. Note min
            // starts with the index of the rightmost
            // hash.
```

```
                for(int i=(r-1)%w; i!=r; i=(i-1+w)%w)

                if (h[i] < h[min]){
                        min = i;
                }
                record(h[min], global_pos(min, r, w));
        }
        else {
        // Otherwise, the previous minimum is still in
        // this window. Compare against the new value
        // and update min if necessary.

                if (h[r] <= h[min]) { // (*)
                        min = r;
                        record(h[min], global_pos(min, r, w));
                }
        }
    }
}
```

6. Avaluació de les eines existents

Dintre la detecció del plagi de codi font existeixen diferents eines, les més conegudes són JPlag i MOSS. A continuació expliquem com obtenir-les, instal·lar-les i utilitzar-les. D'aquesta manera podrem veure pros i contres de cada aplicació i recollir idees per al desenvolupament de la pròpia.

6.1. Registre i instal·lació dels recursos necessaris.

6.1.1. JPLAG

Per instal·lar JPlag[6] primer cal anar a la pàgina oficial.

Entre d'altra informació aquí ens expliquen que per utilitzar l'aplicació s'ha d'omplir un formulari per obtenir un compte d'usuari. En el formulari s'ha d'explicar les nostres intencions d'ús de l'eina així com informar de quina universitat formem part, etc.

Després d'omplir un formulari, ens van respondre amb un usuari i una paraula de pas.
L'accés a l'eina és fa via web a traves de l'explorador[7].

6.1.2. MOSS

Per instal·lar el MOSS[8] s'ha d'anar a la web oficial. En aqueta web ens diuen els passos a seguir per obtenir un compte, s'ha d'enviar un mail a moss@moss.cs.berkeley.edu en el cos del missatge a d'aparèixer el següent:
registeruser
mail *username@domain*

Una vegada enviat el missatge es rep un mail de resposta amb l'script a utilitzar, aquest ja porta el número d'usuari que ens correspon. Així que l'hem de copiar i guardar-lo amb l'extensió *.pl perquè es tracta d'un script en perl.

Si disposem d'un intèrpret de Perl ja podem el podem utilitzar.

6.1.3. SIM

Per instal·lar l'eina SIM[9] només cal anar a la seva web oficial i baixar l'aplicació.

En aquesta eina no ens cal registrar-nos a cap servei ja que l'aplicació s'executa en la màquina local.

---

[6] https://www.ipd.uni-karlsruhe.de/jplag/
[7] https://www.ipd.uni-karlsruhe.de/jplag/start.jnlp
[8] http://www.cs.berkeley.edu/~aiken/moss.html
[9] http://www.cs.vu.nl/~dick/sim.html

6.1.4. YAP

Per instal·lar l'eina YAP[10] s'ha d'anar a la web. Aquí ens podem baixar les tres versions que hi ha d'aquesta aplicació.

Malgrat tot, m'ha estat impossible fer funcionar l'aplicació. Ja que no hi ha cap manual d'instal·lació ni he pogut contactar amb el creador perquè hem dones les explicacions pertinents. És per això que aquesta eina no s'ha pogut utilitzar.

6.2. Funcionament de cada eina.

6.2.1. JPLAG

JPlag (*Bibl. 2*) és una aplicació web per detectar plagis desenvolupada en Java. Suporta múltiples llenguatges com per exemple Java, C i C++, l'algoritme que utilitza és el *Greedy String Tiling*. Aquesta aplicació va ser creada per Guido Malpohl a la universitat alemana de *Karlsruhe*, primer com a treball de recerca i posteriorment s'ha seguit el seu desenvolupament al departament d'informàtica d'aquesta universitat.

Per utilitzar l'aplicació s'ha d'obtenir un usuari i password per tal de poder accedir al servei.

Un cop validats en iniciar l'aplicació per primera vegada ens demana en quin directori volem guardar els resultats, aquests són en format HTML. Seleccionem el directori i idioma i ja accedim al menú principal.

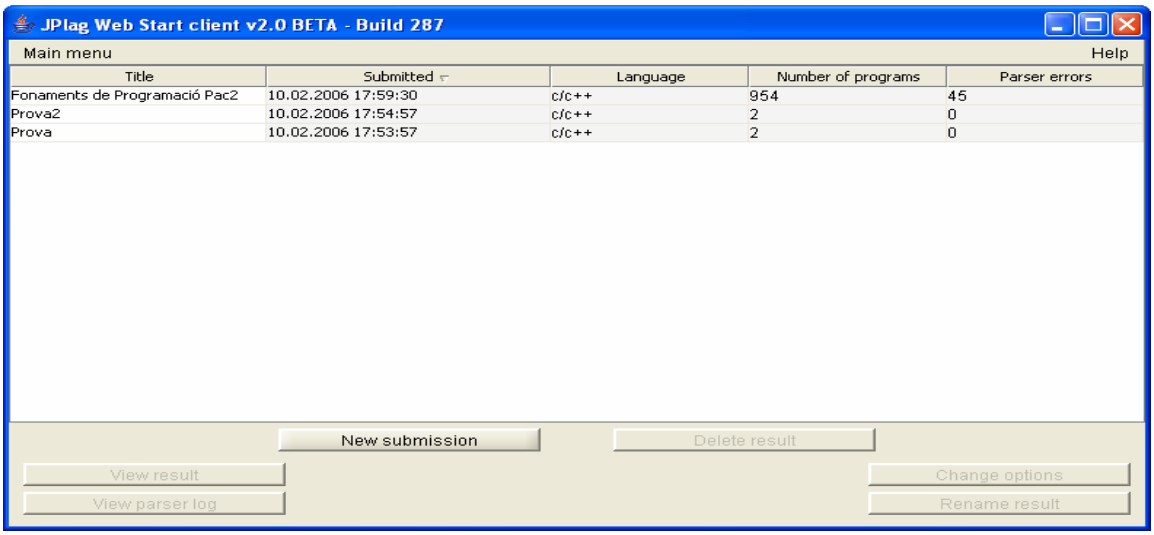


Figura 4 – Menú principal de JPlag

[10] http://luggage.bcs.uwa.edu.au/~michaelw/YAP.html

Un cop validats podem realitzar comparacions a traves de l'opció *New Submission*. Aquesta opció ens mostra un nou menú on indiquem el llenguatge dels documents, el títol de la comparació i el directori on es troben els programes.

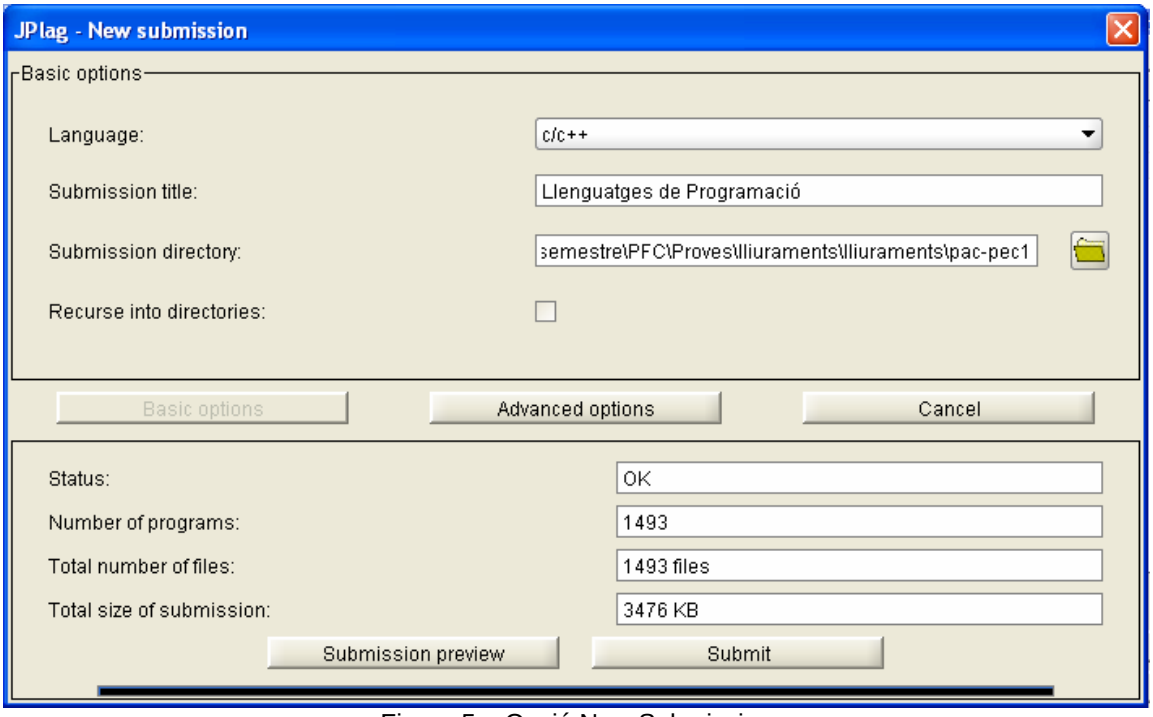


Figura 5 – Opció New Submission

En introduir els mínims paràmetres aquest ens calcula el nombre d'arxius que es compararan així com el seu tamany total. També disposem de l'opció perquè busqui a dintre de directoris.

Una vegada hem executat el procés per processar els programes, ens retorna una pàgina que conté els detalls de l'execució, com es pot veure a la figura 6.

La qual inclou:

- El directori treball on s'han guardat els resultats.
- Nom dels programes comparats.
- Llenguatge de programació seleccionat.
- Extensió dels arxius.
- Nombre de comparacions que es mostren.
- Valor del paràmetre *Minimum Match Length.*

JPlag Search Results

Directory:	C:\Documents and Settings\fixfax\JPlag_Results\prova1
Programs:	Alumne1 - Alumne2
Language:	C/C++ Scanner [basic markup]
Submissions:	2
Matches displayed:	1 (Treshold: 66.6%)
Date:	10.02.2006
Minimum Match Length (sensitivity):	12
Suffixes:	.cpp, .CPP, .c++, .C++, .c, .C, .h, .H, .hpp, .HPP

Distribution:

90% - 100%	0	.
80% - 90%	0	.
70% - 80%	0	.
60% - 70%	1	##
50% - 60%	0	.
40% - 50%	0	.
30% - 40%	0	.
20% - 30%	0	.
10% - 20%	0	.
0% - 10%	0	.

Matches:

Alumne2 -> Alumne1 (66.6%)

Figura 6 – Pàgina principal d'una execució

Dintre la pantalla principal tenim un informe resum sobre en quin percentatge es troben les similituds que s'han trobat.

Segons s'explica a la web[11], els matches segueixen dos criteris d'ordenació, per *average similarity* i per *maximum similarity*.

Average similarity: la semblança mitjana és la mitja de les semblances dels dos programes. Els matches amb una semblança mitja alta indiquen que els programes són similars.

Maximum similarity: la semblança màxima és el màxim de les semblances dels dos programes. Aquesta classificació és molt útil per trobar programes d'estudiants que només han copiat una part del programa d'un estudiant. Tot i que la seva funcionalitat és menor que l'anterior també és important tenir-la en compte per no passar aquests casos per alt.

[11] https://www.ipd.uni-karlsruhe.de/jplag/example/help-sim-en.html

Com veiem a les dos definicions que trobem a la pàgina oficial de JPlag, cada semblança té la seva utilitat. En resum podem dir que la primera semblança serveix per a aquells programes que s'ha produït una copia general i la segona és un cas específic per detectar els estudiants que només copien una part del programa. Aquest segon cas és dona sovint en estudiants que per exemple no sabem con implementar una funció que necessiten per completar la pràctica.

JPlag ens proporciona una interfície per poder comparar el codi dels dos programes, podem navegar per ell i ens ressalta el codi que es coincident. Aquesta interfície és molt útil per veure les parts iguales i considerar si s'ha produït copia o no.

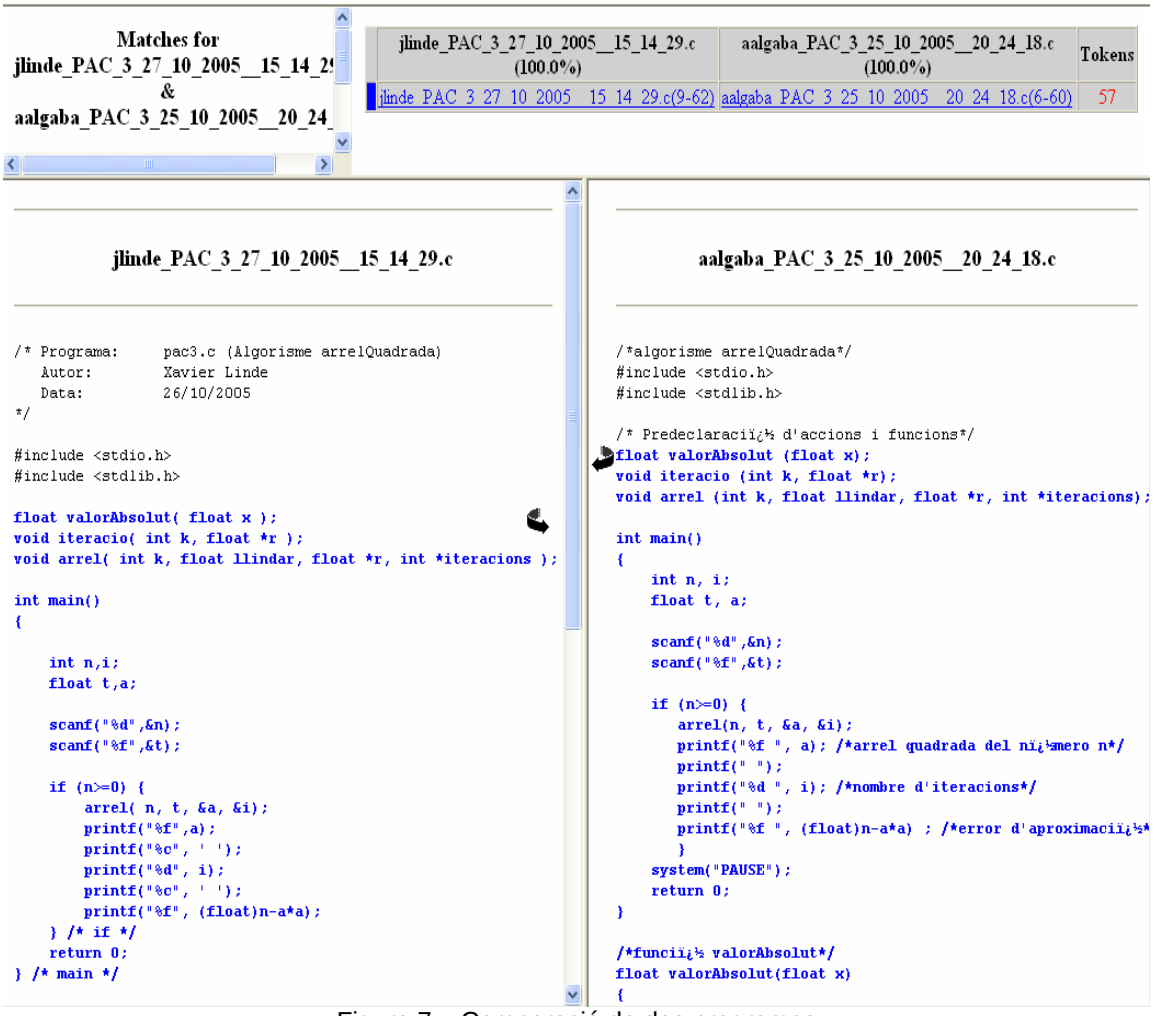


Figura 7 – Comparació de dos programes

6.2.2. MOSS

MOSS (*Bibl. 5*) és una altre eina de detecció de plagi per a múltiples llenguatges de programació com ara: C, C++, Java, Ada, etc. Desenvolupat per *Alex Aiken* al 1994 a la universitat *UC Berkeley*[12]. El nom de l'aplicació ve donat per les seves sigles en angles, *Measure Of Software Similarity*.

La interacció amb l'aplicació es fa des de un script en Perl, existeix versió per a Unix i Windows. Hi ha dos versions de funcionament de l'script, en una envia els documents per correu electrònic al servidor per al seu procés, i la segona envia els arxius amb una connexió directe.

En funció de les condicions de la xarxa on volem executar l'eina es fa servir un o altre.

A continuació mostrem un exemple d'execució:

```
E:\UOC\Quart semestre\PFC\Teoria\Eines de plagi\MOSS\Eina>perl mosswin.pl -l c test1.c test2.c
Checking files . . .
OK
Uploading test1.c ...done.
Uploading test2.c ...done.
Query submitted.  Waiting for the server's response.

http://moss.cs.berkeley.edu/~moss/results/201283199

Will try and open the above URL...

E:\UOC\Quart semestre\PFC\Teoria\Eines de plagi\MOSS\Eina>_
```

Figura 8 – Exemple d'execució MOSS

Com podem veure a l'exemple d'execució l'script primer fa una comprovació dels arxius, si aquests són correctes els envia al servidor on es troba l'aplicació. El servidor ens retorna un *link*, en aquesta pàgina web es troben els resultats de la comparació.

Moss Results

Tue Apr 18 08:27:38 PDT 2006

Options -l c -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
test1.c (82%)	test2.c (82%)	28

Any errors encountered during this query are listed below.

Figura 9 – Resultat de l'execució MOSS

[12] http://www.berkeley.edu/

En aquesta pàgina ens mostren els parells de documents que s'han comparat i ens informa del nombre de línees coincidents així com del percentatge de similitud entre ells.

Aquesta eina també ofereix navegar en detall per el codi dels programes comparats, de la mateixa forma que ho fèiem amb JPlag.

Figura 10 – Navegació per el codi de la comparació

L'eina efectua optimitzacions sobre alguns dels algoritmes més usuals. L'autor no especifica quines són aquestes optimitzacions.

A la pàgina oficial trobem un enllaç[13] on es mostren les idees principals del funcionament de l'algorisme, per tant es dedueix que utilitza l'algorisme WINNOWING i és en aquest on implementa les optimitzacions.

[13] http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf

6.2.3. SIM

Aquesta eina està desenvolupada en C (*Bibl. 6*). Com les altres permet trobar similituds entre programes escrits en *C, Java, Pascal, Modula-2*, etc. Ha estat creada per *Dick Grune* i és l'eina que utilitzen per comparar les pràctiques dels seus estudiants. El fet diferencial de les altres aplicacions es que no és client servidor, s'executa únicament a la màquina client.

L'objectiu de l'eina com de totes les altres és veure quina parts dels programes són similars. Considera que dos parts dels programes són similars si únicament és diferencien en un *layout*, comentaris, identificadors i continguts numèrics, cadenes i caràcters.

Una vegada baixats els binaris, s'observa que cada llenguatge s'associa a un executable. Per el llenguatge C és *sim_c.exe*, per a Java *sim_java.exe*, etc. En el nostre cas fem servir *sim_c.exe*.

A continuació mostrem un exemple d'execució:

```
E:\UOC\Quart semestre\PFC\Teoria\Eines de plagi\SIM\Eina>sim_c.exe aalgaba_PAC_1_28_09_2005__19_37_13.c
File aalgaba_PAC_1_28_09_2005__19_37_13.c: 122 tokens
Total: 122 tokens


E:\UOC\QUARTS~1\PFC\Teoria\EINESD~1\SIM\Eina>sim_c.exe test1.c test2.c
File test1.c: 165 tokens
File test2.c: 165 tokens
Total: 330 tokens

test1.c: line 3-40                    |test2.c: line 3-40              [165]
#define MAX 100                       |#define MAX 100
                                      |
int main( int argc, char **argv )     |int main( int argc, char **argv )
{                                     |{
int i, j;                             |int i, j;
int prime[ MAX ];                     |int prime[ MAX ];
                                      |
if ( argc != 1 ) {                    |if ( argc != 1 ) {
fprintf( stderr, "%s: no command line |fprintf( stderr, "%s: no command line
exit ( 1 );                           |exit ( 1 );
}                                     |}
                                      |
for( i = 2; i < MAX; i++ ) {          |for( i = 2; i < MAX; i++ ) {
prime[ i ] = 1;                       |prime[ i ] = 1;
}                                     |}
                                      |
                                      |
/* Erathostenes prime number sieve */ |/* Erathostenes prime number sieve */
for( i = 2; i < MAX; i ++ ) {         |for( i = 2; i < MAX; i ++ ) {
if ( prime[ i ] ) {                   |if ( prime[ i ] ) {
for ( j = i + i; j < MAX; j += i ) {  |for ( j = i + i; j < MAX; j += i ) {
 prime[ j ] = 0;                      | prime[ j ] = 0;
}                                     |}
}                                     |}
}                                     |}
                                      |
/* Print prime numbers */             |/* Print prime numbers */
i = MAX;                              |i = MAX;
fprintf( stdout, "Prime numbers betwee |fprintf( stdout, "Prime numbers betwee
for( i = 2; i < MAX; i++ ) {          |for( i = 2; i < MAX; i++ ) {
if ( prime[ i ] ) {                   |if ( prime[ i ] ) {
fprintf( stdout, "%d ", i );          |fprintf( stdout, "%d ", i );
}                                     |}
}                                     |}
fprintf( stdout, "\n" );              |fprintf( stdout, "\n" );
                                      |
return 0;                             |return 0;
}                                     |}


E:\UOC\QUARTS~1\PFC\Teoria\EINESD~1\SIM\Eina>_
```

Figura 11 – Exemple d'execució SIM

Com a resultat ens mostra el nombre de tokens de cada arxiu i el seu total. De cada bloc que considerar igual ens diu en quines línees es troba en el fitxer font. A més mostra directament el codi dels dos arxius.

Com podem observar el format de la sortida no està tant treballat com en les eines anteriors, segons comenten a la web s'han implementat scripts per a tractar-la i fer-la més amable a través de la creació d'histogrames i resums que ens duran a una millor interpretació de la sortida.

6.3. Jocs de proves.

Per realitzar els jocs de proves es faran servir les PAC's i pràctiques de l'assignatura Fonaments de Programació.

Aquesta assignatura està formada per 4 PAC's i una pràctica, dintre cada carpeta tenim totes les entregues tant parcials com finals de cada PAC.

En aquesta assignatura en concret els estudiants fan servir un corrector automàtic per comprovar per comprovar les entregues. És per això que dintre de cada directori que conforma una entrega tenim les entregues finals i parcials que ha realitzat cada alumne, és a dir que ens trobem amb entregues que no compilen, entregues incompletes, arxius que no són de text pla, etc.

De totes maneres farem servir tot aquest conjunt d'entregues per realitzar les proves ja que serà l'entrada de l'aplicació que s'ha de desenvolupar.

Les proves s'han realitzat des d'un ordenador PC amb les següents característiques:

- Processador AMD 64bits 3200.
- 512 Mb de RAM.
- Windows XP.
- Ample de banda: 1Mb/s

Cal tenir en compte que el procés dels fitxers en les eines JPlag i MOSS es fan en els servidors que tenen dedicats.

Nombre de programes de cada entrega:

Fonaments de Programació	**Nº entregues**
Pac1	1629
Pac2	1174
Pac3	1080
Pac4	1461
Pràctica	1697
Total:	7041

6.3.1. JPLAG

La primera aplicació que provarem serà JPlag. La seva interfície és molt senzilla i intuïtiva. Per cada pràctica creo una New submission.

Les proves es fan amb Minimum Match Length: 12

Fonaments de Programació	Temps d'execució	Prog. Processats	Prog. Error
Pac1	2 minuts 11 segons	1545	84
Pac2	1 minut 52 segons	1129	45
Pac3	1 minut 37 segons	1046	34
Pac4	2 minuts 8 segons	1439	22
Pràctica	3 minuts 36 segons	1675	20

Per cada execució es mostra el gràfic de similituds.

Pac1:

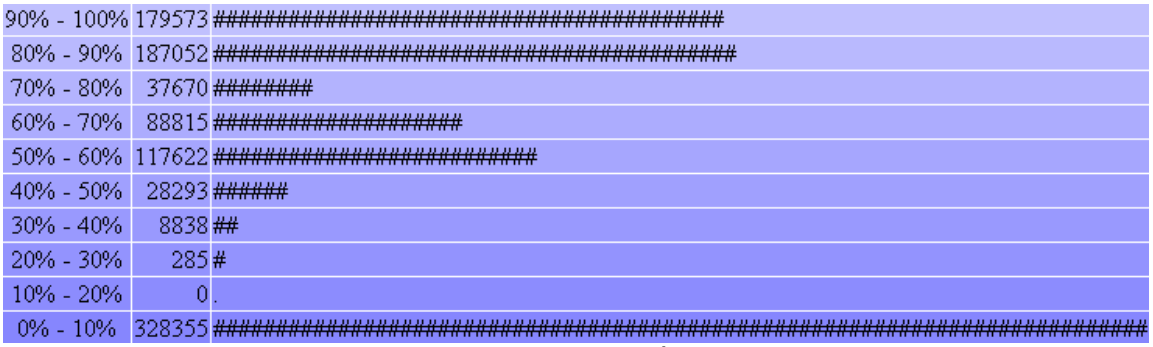


Figura 12 – Distribució PAC1

Pac2:

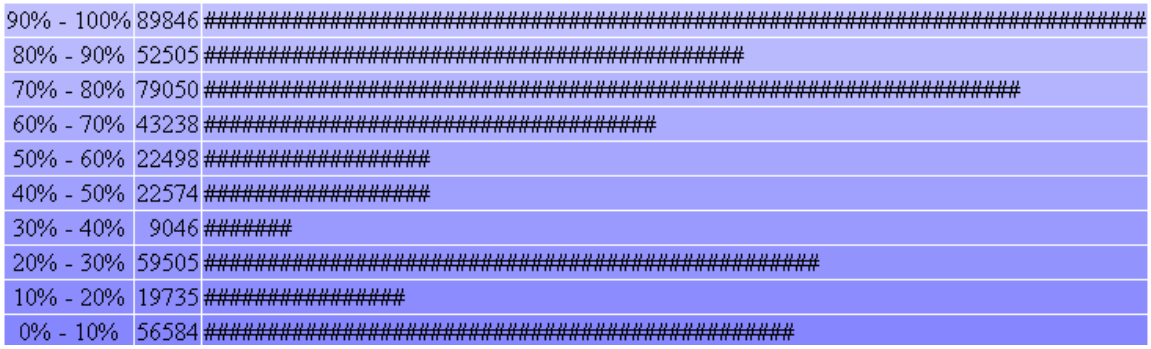


Figura 13 – Distribució PAC2

Pac3:

90% - 100%	72275	###########################
80% - 90%	26780	##########
70% - 80%	47870	##################
60% - 70%	32336	############
50% - 60%	30013	###########
40% - 50%	21139	########
30% - 40%	15526	######
20% - 30%	16885	######
10% - 20%	1002	#
0% - 10%	191709	##

Figura 14 – Distribució PAC3

Pac4:

90% - 100%	217457	##
80% - 90%	137048	##
70% - 80%	92226	################################
60% - 70%	71047	#########################
50% - 60%	55118	###################
40% - 50%	34740	###########
30% - 40%	17927	######
20% - 30%	10134	###
10% - 20%	4155	#
0% - 10%	160893	##

Figura 15 – Distribució PAC4

Pràctica:

90% - 100%	523884	##
80% - 90%	311934	###
70% - 80%	160718	######################
60% - 70%	68997	#########
50% - 60%	28021	####
40% - 50%	6870	#
30% - 40%	9097	#
20% - 30%	5512	#
10% - 20%	14882	##
0% - 10%	13913	#

Figura 16 – Distribució Pràctica

Com podem observar en els gràfics, el nombre de comparacions que s'han de realitzar amb l'entrada que em proporcionat a l'eina és molt elevat, així com el nombre de resultats que sobrepassen el 80% d'igualtat.

Després de les proves realitzades s'observa que JPlag segueix detectant copies encara que es produeixin les següents modificacions:

- Format del document: afegir línees en blanc, comentaris a qualsevol part del document, espais, etc.
- Modificacions dels noms de les variables o funcions.
- Modificacions en valors constants.

Modificacions que es poden realitzar i no detecta JPlag:

- Combinar sub expressions en una de sola o fragmentar càlculs amb l'ús de variables temporals.
- Afegir mètodes o variables que no s'utilitzen.

6.3.2. MOSS

A l'hora d'executar el joc de proves, el primer problema que m'he trobat és que l'script abans d'enviar els arxius del servidor comprova, entre d'altres coses, que es tracti d'un arxiu de text pla. Com s'ha comentat, dintre les carpetes hi ha arxius que no són de text pla perquè un alumne ha fet una entrega errònia. És per això que he modificat el *script* perquè esborri els arxius que no són de text pla i així ja puc passar el joc de proves.

Fonaments de Programació	Temps d'execució	Prog. Processats	Prog. Esborrat
Pac1	4 minuts 1 segon	1550	79
Pac2	1 minut 46 segons	1132	42
Pac3	1 minut 40 segons	1053	27
Pac4	4 minuts 48 segons	1445	16
Pràctica	6 minuts 33 segons	1682	15

Després de les proves realitzades s'observa que MOSS segueix detectant copies encara que es produeixin les següents modificacions:

- Ordre d'aparició de les funcions.
- Format del document: afegir línees en blanc, comentaris a qualsevol part del document, espais, etc.
- Modificacions dels noms de les variables o funcions.
- Modificacions en valors constants.

La principal diferència entre JPlag i MOSS radica en l'algorisme que utilitzen. En el cas de MOSS únicament necessita un analitzador lèxic i no un *parser* com JPlag, és per això que MOSS pot processar arxius que no compilen i en canvi JPlag només els processa fins al lloc on dona l'error de compilació.

6.3.3. SIM

Per realitzar les proves amb aquesta eina no s'ha tingut que esborrar cap arxiu. A més no ha donat cap error a l'hora de processar els arxius que no són de text pla. De manera que ens els resultats ens trobem amb comparacions d'arxius que no caldria fer-les ja que no corresponen a codi C, aquestes sortides dificulten la lectura dels resultats.

Fonaments de Programació	Temps d'execució	Prog. Processats	Prog. Error
Pac1	19 segons	1629	0
Pac2	7 segons	1174	0
Pac3	3 segons	1080	0
Pac4	5 segons	1461	0
Pràctica	8 segons	1697	0

Com es pot observar el temps d'execució és menor comparat amb les altres eines, això és degut perquè no s'han de transferir arxius ni resultats.

Degut a la presentació dels resultats, aquesta eina es desaconsella si s'han de processar un nombre molt elevat de programes, ja que no disposa d'una interfície amigable per analitzar els resultats que s'han produït.

6.4. Anàlisis de les eines.

Després d'utilitzar les diferents eines considero que en termes d'us JPlag és superior a les altres, la seva interfície gràfica està més treballada i per tant proporciona una millor interacció. A més es descarrega els resultats al propi PC, així que els tindrem sempre a disposició.

Excepte JPlag les altres eines no fan un resum amb la distribució del percentatge. Com ja s'ha comentat aquesta dada es considera de gran utilitat per obtenir una primera visió dels resultats que s'han produït després d'executar el procés, i més encara quan es tracta d'un nombre molt elevat de pràctiques. Ja que amb el gràfic l'usuari sap en quin percentatge s'han detectat el major nombre de comparacions.

A més en les eines MOSS i SIM el resultat de les comparacions no segueix cap ordenació, quan la comparació és realitza en un gran nombre de pràctiques és molt important seguir algun criteri d'ordenació per presentar les comparacions ja que sinó és fa del tot inintel·ligible per l'usuari.

En termes deficiència considero que amb el MOSS pots obtenir millor resultats. Sobretot per la implementació que fa de detectar parts de codi que són comunes a totes les pràctiques i així no tenir-les en compte a l'hora de calcular la seva similitud.

Com ja s'ha comentat l'eina SIM tot i tenir els millors temps d'execució, perquè l'execució és en local i no s'han de transferir els arxius, proporciona una sortida que dificulta molt a l'usuari la lectura dels resultats.

En general considero que la representació del informe general dels resultats que s'han produït és millorable en totes les eines.

7. Anàlisi de l'eina automàtica de plagis

Fins aquest punt s'ha estat explicant en que consisteix el plagi, els algoritmes de detecció i s'han analitzat els programes que existeixen.

En aquest capítol s'ha de treure profit de tota aquesta informació per analitzar els requeriments de l'eina de plagi.

7.1. Requeriments per al desenvolupament i integració de l'eina a l'entorn UOC.

7.1.1. Característiques principals

Després d'analitzar els requeriments necessaris que ha de complir l'eina es considera que les principals característiques que ha de tenir són: facilitat d'ús, ràpida execució, fiabilitat i claredat al mostrar els resultats.

- Facilitat d'ús: tot i que serà un usuari expert l'encarregat d'interactuar amb l'aplicació, ja que el corrector és d'assignatures d'informàtica. Pel tipus d'aplicació aquesta no ha de tenir interfície complexa, tenint en compte el tipus d'aplicacions existents a la UOC el més convenient serà desenvolupar una aplicació web.
  Des d'aquesta interfície l'usuari podrà seleccionar les opcions d'execució.

- Ràpida execució: necessitem que el temps de resposta de l'aplicació sigui bo, ja que sinó podem provocar la frustració de l'usuari. Com a referència prendrem els temps que s'han obtingut de les execucions de les aplicacions web JPlag i MOSS.
  L'algorisme que utilitzarem serà *Greedy String Tiling*, ja que ofereix una bona relació entre rapidesa i fiabilitat dels resultats.

- Fiabilitat: l'objectiu final de l'eina és reduir al mínim el temps que ha dedicar un professor per detectar casos de plagi en les entregues que realitzen els seus estudiants, així com millorar la detecció d'aquestes.
  Evidentment per no frustrar a l'usuari és molt important que l'eina no detecti casos falsos de còpia, ja que llavors perdrem la confiança d'aquest i podem fer que hagi de dedicar més temps al procés de detecció de plagis.
  Com s'ha comentat al llarg de la memòria, en algunes assignatures d'informàtica és dona part del codi per estructurar la pràctica. En aquests casos ens trobarem amb fitxers que contenen parts de codi iguals, però no volen dir que s'hagi produït un plagi. És important que donem l'opció a l'usuari d'introduir el fitxer amb el codi inicial que parteixen els alumnes per no tenir-lo en compte.

- Claredat: com s'ha comentat, si en algun punt es considera que fallen les aplicacions avaluades és en la presentació dels resultats. De manera que és important que no caiguem en el mateix error. Com a resum, les principals dades a mostrar són: nombre d'arxius processats, nombre d'arxius que no s'han pogut processar, gràfic resum de les semblances obtingudes, mostra de les comparacions ordenades de major a menor semblança i en segon lloc que l'usuari pugui entrar a veure el codi ressaltat de les comparacions que ell desitgi

7.1.2. Eliminació de l'enunciat

Com ja s'ha comentat, algunes vegades el professor dona l'estructura del programa. Per no tenir en compte les parts de codi ja donades es seguiran els següents passos:

- Traduir a tokens el programa que conté l'estructura inicial.
- Aplicar l'algorisme *GST* de cada programa amb el programa que conté l'estructura inicial. Els tokens que es troben en cada *match* es marquen amb un valor especial, aquest valor significa que aquets tokens no es tindran en compte a l'hora de trobar matches entre els programes.
- Aplicar l'algorisme GST als programes que es volen comparar, tenint en compte que el tokens amb el caràcter especial no s'han de tenir en compte.

D'aquesta forma donem l'opció a l'usuari que pugui introduir el codi que no vol que es tingui en compte a l'hora de fer les comparacions.

7.2. Anàlisi funcional

7.2.1. Definició de les paraules reservades

El primer pas per aplicar l'algorisme consisteix en fer la traducció del codi C a *tokens*. A continuació mostro la traducció inicial que es farà, en base de l'experiència en un futur es poden introduir modificacions.

Llenguatge C	**Token**	**Enter**
`#include <stdio.h>`	`Library`	`2`
`#define FINAL 0`	`DefineConstant`	`24`
`int main() {`	`BeginFunction`	`47`
`}`	`EndFunction`	`1`
`int n;`	`DefVar`	`3`

`scanf("%d", &n);`	CallFunction	4
`p=0;`	Assign	12
`while (n!=0) {`	BeginWhile	44
`}`	EndWhile	36
`for (i=1; i<=p; i++) {`	Assign BeginFor	12 46
`}`	EndFor	38
`do {`	BeginDoWhile	45
`}while(n==10);`	EndDoWhile	37
`return 0;`	Return	43
`if(i==0){`	BeginIf	48
`}`	EndIf	49
`else{`	BeginElse	50
`}`	EndElse	34
`switch( Grade ) {`	BeginSwitch	51
`case 'A' : printf( "Excellent" );`	BeginCase	52
	EndCase	31
`case 'B' : printf( "Good" );`	BeginCase	52
	EndCase	31
`case 'C' : printf( "OK" );`	BeginCase	52
	EndCase	31
`default : printf( "Bad" );`	BeginDefault	53
	EndDefault	32
`}`	EndSwitch	35
`break;`	Break	42
`goto etiqueta1;`	Goto	40
`continue;`	Continue	41
`n /= b;`	Assign Div	12 8
`n %= b;`	Assign Mod	12 9
`n += b;`	Assign Add	12 10
`n -= b;`	Assign Sub	12 11
`n <<= b;`	LeftAssign	18
`n >>= b;`	RightAssign	19
`n &= b;`	AndAssign	20

```
n ^= b;                                XorAssign              21

n |= b;                                OrAssign               22

n = p / b;                             Assign Div             12 8

n = p * b;                             Assign Mul             12 7

n = p % b;                             Assign Mod             12 9

n = p + b;                             Assign Add             12 10

n = p - b;                             Assign Sub             12 11
n = p * b/r;                           Assign Mul Div         12 7 8

typedef enum {FALSE,TRUE} bool;        Typedef Enum           23 30

typedef struct                         Typedef Struct         23 27
{                                      BeginStruct            26
   int  id;                            VarStruct              29
   bool esPortatil;                    VarStruct              29
   int  unitats;                       VarStruct              29
} tProducte;                           EndStruct DefVar       25 3
```

Amb aquesta traducció, no tindrem en compte el format del document, ja siguin línees en blanc, comentaris, espais, etc.
Tampoc es consideren els noms de les variables o funcions, ni dels valors que prenen les constants.

Per realitzar aquesta traducció el *parsing* dels fitxers es farà amb les eines JLex (*Bibl. 7*) i JCup (*Bibl. 8*). Com podem veure a la figura 3 amb JLex es fa la definició de les paraules reservades de la gramàtica i amb JCup definim la gramàtica del llenguatge, en aquest cas es tracta de ANSI C.

Durant l'anàlisi d'un fitxer de codi font, quan una seqüència de paraules reservades encaixa amb una regla de la gramàtica és quan es pot donar d'alta un Token.

Prenem com a base el compilador *gcc*[14] ja que és el que s'utilitza per corregir les pràctiques dels estudiants. A la pràctica això significa que qualsevol fitxer C que pugui compilar *gcc* l'eina ha de ser capaç de realitzar el *parsing* per generar la seva corresponent traducció a tokens.

[14] http://gcc.gnu.org

7.2.2. Interfície de l'eina de plagis.

La interacció de l'usuari amb l'aplicació serà via web. És defineixen les següents pantalles:

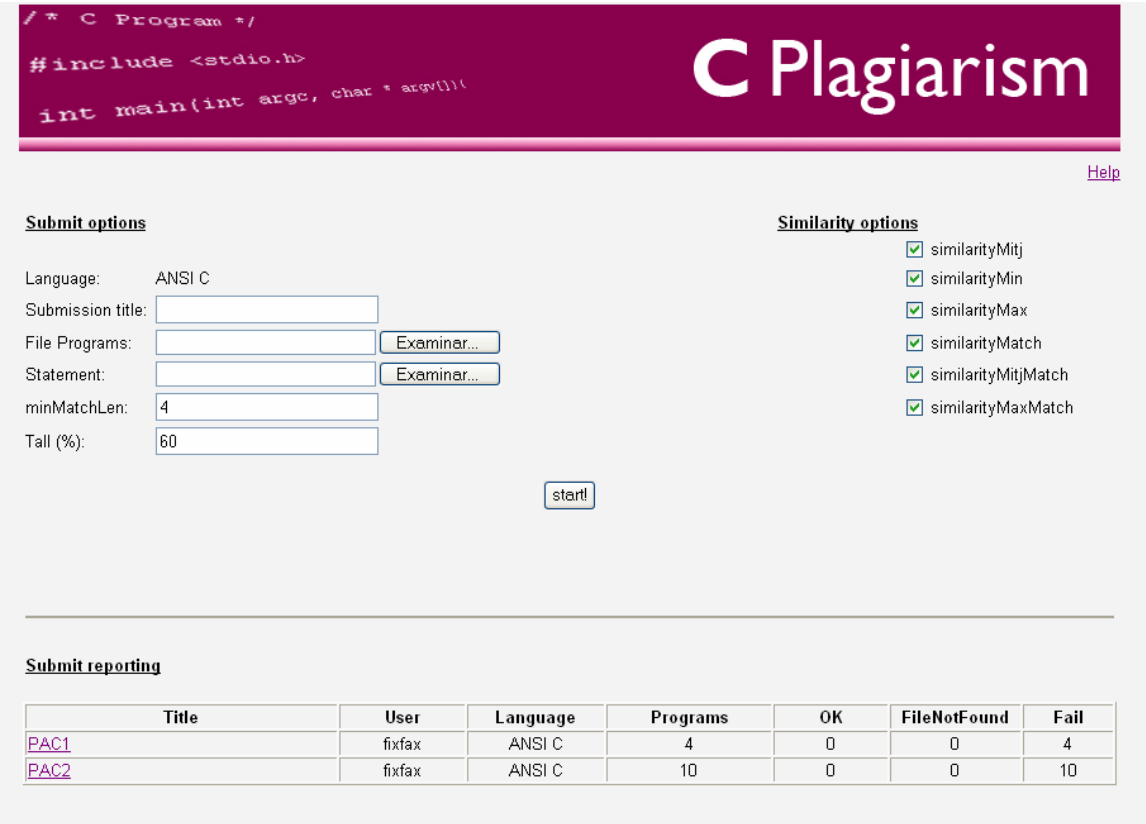


Figura 17 – Pantalla inicial de l'aplicació

Dintre la pantalla principal es defineixen tres seccions:

- *Submit options*: en aquesta secció s'especifica el llenguatge dels programes i el títol de la comparació que es realitza. A continuació s'ha d'especificar l'arxiu **.zip* que conté els programes que es volen comparar. Únicament es llegiran els arxius amb extensió *.c . La casella *Statement* serveix perquè l'usuari indiqui l'arxiu *statement.c* que conté el codi que no vol que es tingui en compte en les comparacions. La casella minMatchLen especifica la longitud mínima dels matches. La casella Tall indica el percentatge mínim que han de tenir les comparacions perquè s'incloguin a la pantalla que mostra els resultats.
- *Similarity options*: aquesta secció serveix perquè l'usuari indiqui quines mesures de similitud vol que es calculin. La descripció de les fórmules de cada mesura s'explica a la pantalla *help*.
- *Submit reporting*: històric amb les comparacions que ha realitzat un usuari. D'aquesta forma sempre es podrà accedir als resultats de comparacions antigues.

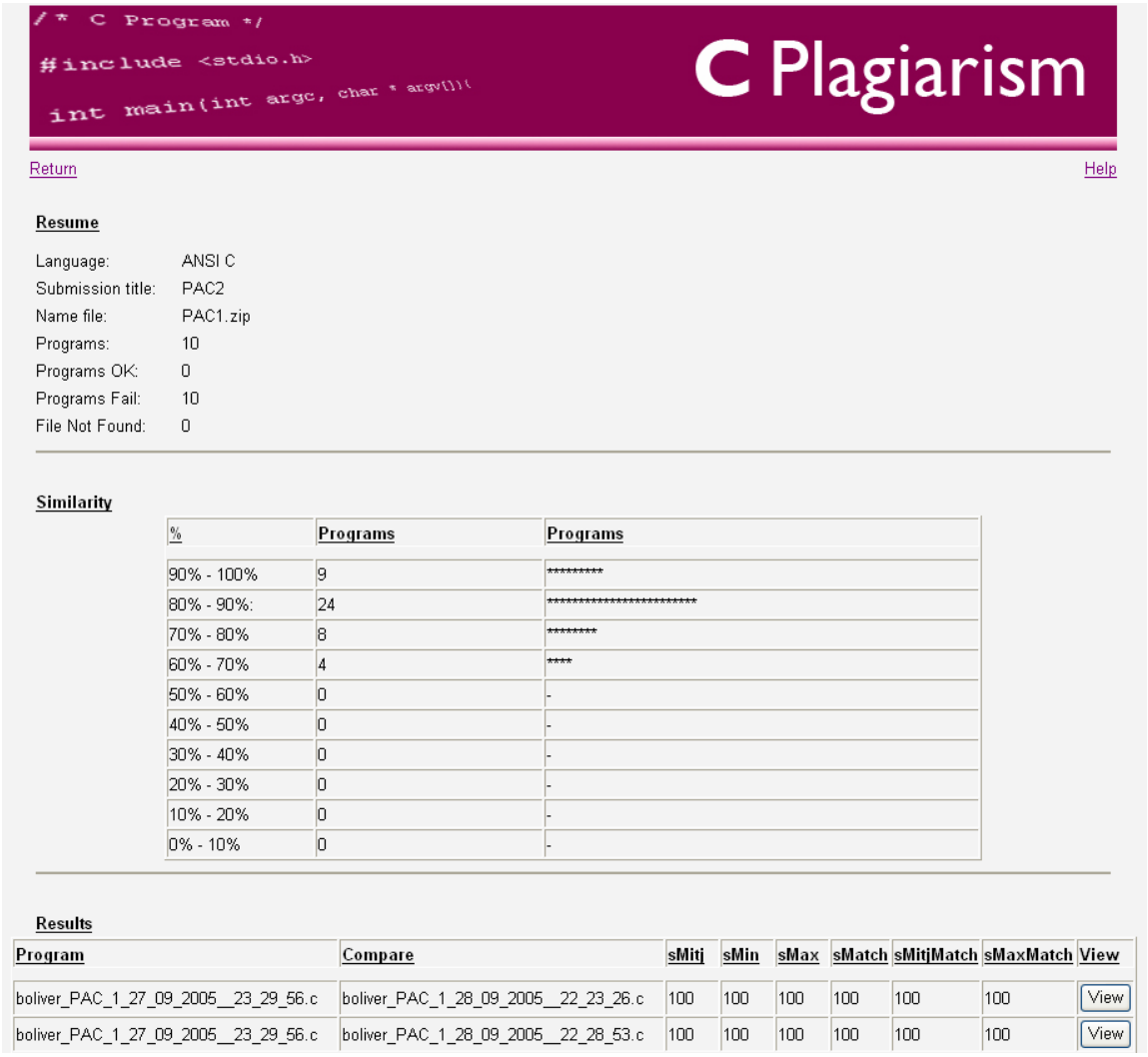


Figura 18 – Pantalla amb els resultats d'una comparació

La figura 18 mostra els resultats d'una comparació. També es diferencien tres seccions:

- *Resume*: ens mostra un resum dels resultats de l'execució. Llenguatge dels programes, el títol de la comparació, l'arxiu que s'ha processat que conté els programes, el nombre de programes que hi ha, els programes que s'han pogut processar, el nombre de programes que no s'han pogut processar i els que no s'han trobat.
- *Similarity*: aquesta secció ens mostra el gràfic de similituds, la intenció d'aquest és que l'usuari pugui identificar de forma ràpida en quin rang s'han trobat el major nombre de comparacions. El gràfic està format per el rang de tant per cent, el nombre de programes que es troben en aquest percentatge i a la tercera columna es realitza un simple gràfic de barres.

- *Results*: es mostra a l'usuari una taula amb les comparacions que s'han realitzat, aquesta estarà ordenada per la similitud *sMitj* de major a menor. La taula únicament conté les comparacions que han donat un percentatge superior al especificat per l'usuari en la pantalla inicial.
  Des de la mateixa taula es dona l'opció a l'usuari perquè pugui veure en detall els matches que s'han trobat en la comparació.

bperea_PAC_1_29_09_2005__19_51_45.c	brus_PAC_1_29_09_2005__19_52_37.c	Tokens
bperea_PAC_1_29_09_2005__19_51_45.c[8-15]	brus_PAC_1_29_09_2005__19_52_37.c[6-13]	8
bperea_PAC_1_29_09_2005__19_51_45.c[16-21]	brus_PAC_1_29_09_2005__19_52_37.c[14-19]	10
bperea_PAC_1_29_09_2005__19_51_45.c[22-26]	brus_PAC_1_29_09_2005__19_52_37.c[20-24]	6

Return

bperea_PAC_1_29_09_2005__19_51_45.c

```
0:      /* Codificacio de l'algorisme Canvi de Base */
1:      /* Benjami Perea Manera*/
2:
3:      #include <stdio.h>
4:      #include <stdlib.h>
5:
6:      int main ()
7:      {
8:        int n, b, p, r, c, i, exp;  /* declaracio de variables */
9:        scanf ("%d", &n);
10:       while (n>0) {
11:             scanf ("%d", &b);
12:             p=0;
13:             r=0;
14:             while (n>0) {
15:                   c=n-(b*(n/b));
16:                   n=(n/b);
17:                   exp=1 ;
18:                   for (i=1; i<=p; i++) {
19:                       exp=(10*exp);
20:                       }
21:                   r=(r+c)*exp;
22:                   p=p+1 ;
23:                   }
24:             printf ("%d ",r);
25:             scanf ("%d", &n);
26:             }
27:
28:       /*system("PAUSE");*/
29:       return 0;
30:     }
```

brus_PAC_1_29_09_2005__19_52_37.c

```
0:      #include <stdio.h>
1:      #include <stdlib.h>
2:
3:      int main(canviBase)
4:      {
5:      int n, b, p, r, c;
6:      int i, exp;
7:      scanf("%d", &n);
8:      while (n!=0){
9:            scanf("%d", &b);
10:           p=0;
11:           r=0;
12:           while (n!=0){
13:                 c=n%b;
14:                 n=n/b;
15:                 exp=1;
16:                 for (i=1; i<=p;i++){
17:                     exp=10*exp;
18:                     }
19:                 r=r+c*exp;
20:                 p=p+1;
21:                 }
22:           printf("%d", r);
23:           scanf("%d", &n);
24:           }
25:           system("PAUSE");
26:           return 0;
27:     }
```

Figura 19 – Detall d'una comparació

La figura 19 és un exemple del detall que es mostra a l'usuari quan decideix veure el codi dels programes d'una comparació. A la part superior es mostra una taula amb els matches que s'han trobat, indicant les línees en que es troben i el nombre de tokens. A cada match se li assigna un color. A la part inferior de la pantalla es mostra el codi amb els colors corresponents a cada match i el número de línea.

Per últim des de la pantalla principal i de resultats es podrà accedir a una pantalla d'ajuda on es mostrarà un resum de les fórmules que s'utilitzen per calcular les similituds i altres indicacions que es considerin d'interès.

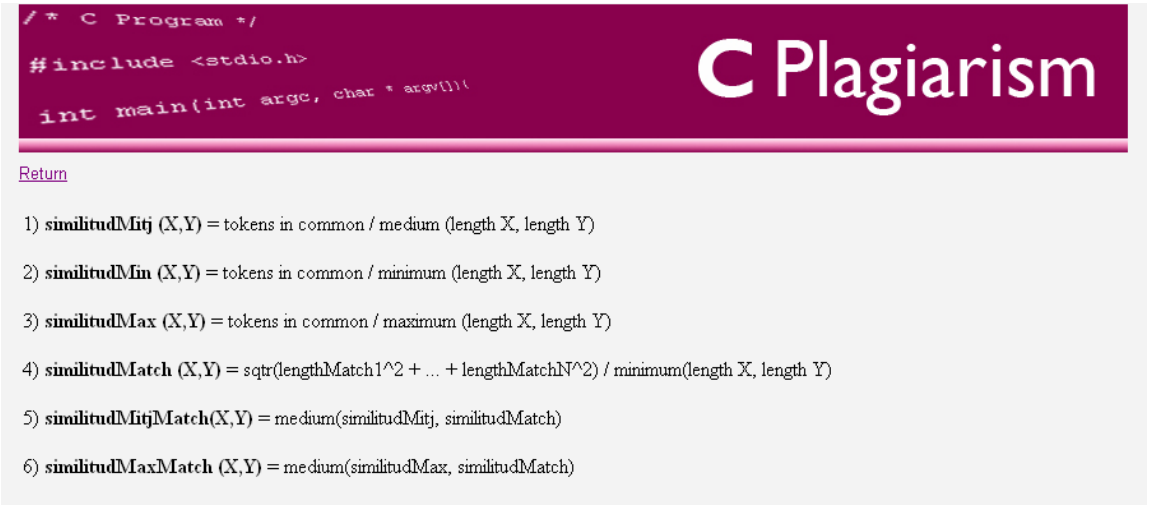


Figura 20 – Pantalla d'ajuda

7.2.3. Diagrama de classes.

Com s'ha comentat per crear el motor de l'aplicació s'utilitzen les eines *JLex* i *JCup*, el llenguatge que s'utilitza per desenvolupar l'aplicació és *Java* (*Bibl. 9*). L'aplicació estarà formada per el següent diagrama de classes, figura 20.

Les classes que formen el diagrama són:

- *JSP*: amb aquesta classes es programaran les pàgines JSP que permetran a l'usuari interactuar amb l'aplicació.
- *Servlet*: es crea un servlet que serà l'encarregat de controlar el flux de l'aplicació. Aquest interactua amb el motor de plagi i redirecciona a la vista adequada.
- *JLex & JCup*: Amb aquestes classes es realitza la definició de les paraules reservades i la gramàtica del programa, per generar la traducció a Tokens.
- *Token*: classe encarregada de modelar un token. Cada programa es tradueix a un conjunt de tokens, guardem la seva representació en un enter, a més tenim l'opció de marcar aquest token per quan apliquem l'algorisme *GST*.
- *TSimbol*: classe encarregada de modelar el conjunt de tokens que forma un programa C.
- *TProgram*: classe encarregada d'emmagatzemar tots els programes que s'han parcejat, conté el mètode per fer la comparació i aplicar l'algorisme *GST* entre el programes parcejats.
- *Tile*: Modela un match d'un programa.
- *ResultParser*: emmagatzema el resultat d'una comparació.
- *ResultsParser*: emmagatzema els resultats de totes les comparacions que s'han realitzat, a més conté mètodes per mostrar un resum.
- *Similarity*: Classe que calcular les mitjanes de similitud.
- *Compare*: aquesta classe conté el noms dels programes que s'han comparat així com els resultats de les similituds.
- *Graphic*: Aquesta classe realitza els càlculs per mostrar el gràfic resultant de la comparació.
- *PropertiesStore*: classe per accedir a les variables del fitxer config.properties.

Diagrama de classes

Capa de presentació

Capa de negoci

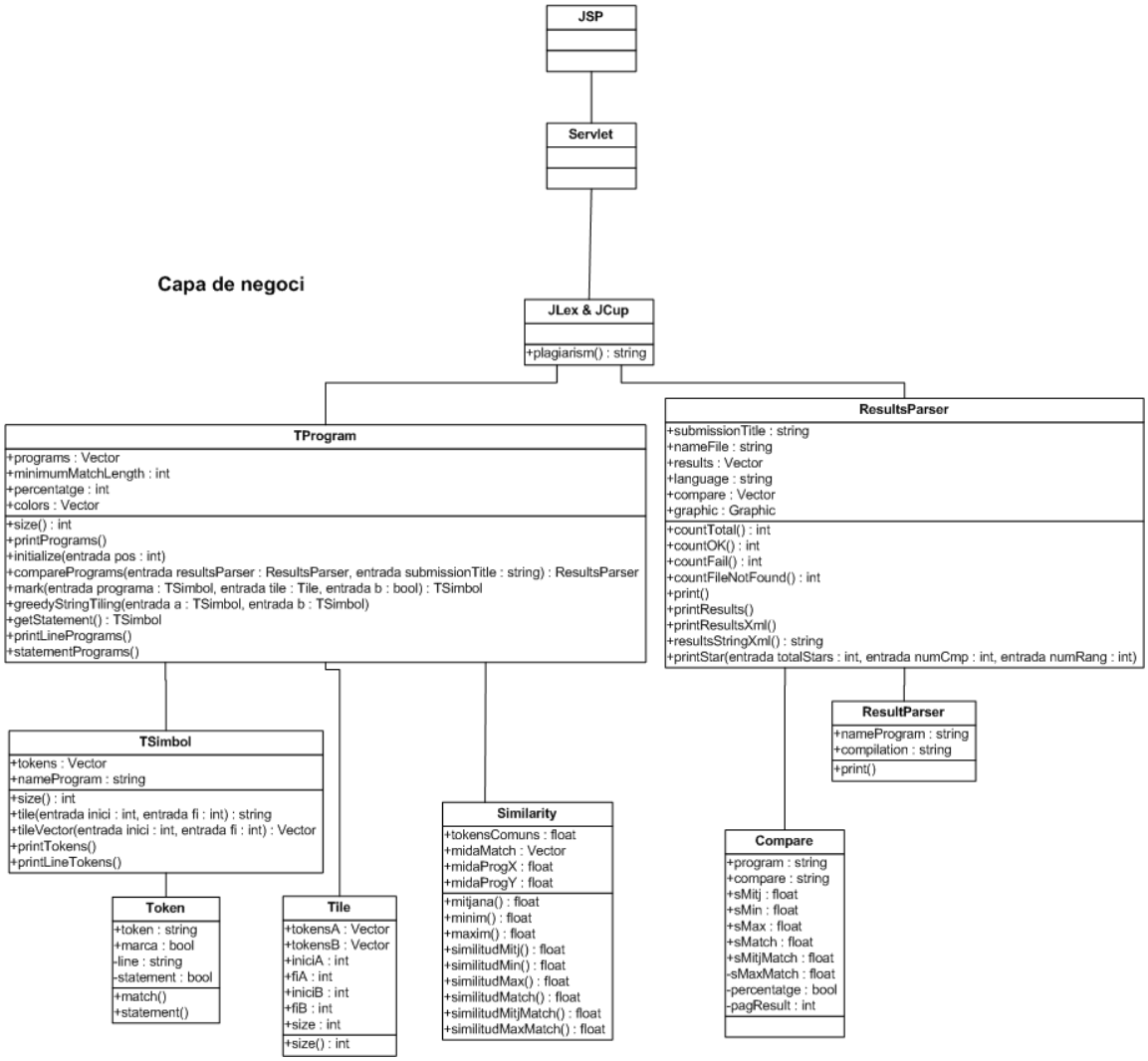


Figura 21 – Diagrama de classes

7.2.4. Funcionament intern.

A continuació s'explica el funcionament de l'eina fent un seguiment d'execució posant especial èmfasi a les funcions principals.

Partim del fitxer plagi.jsp, aquest modela la pàgina inicial de l'eina on l'usuari pot introduir els paràmetres per fer la comparació. La pàgina envia els paràmetres al servlet, que s'encarrega de pujar els arxius al servidor i recuperar els paràmetres d'execució per fer la crida a la funció de plagi.

Un cop cridem a la funció de plagi aquesta ens retorna en un String el XML (*Bibl. 10*) de l'execució, el servlet s'encarrega de processar aquest XML per aplicar el XSL i així mostrar la pantalla amb els resultats.

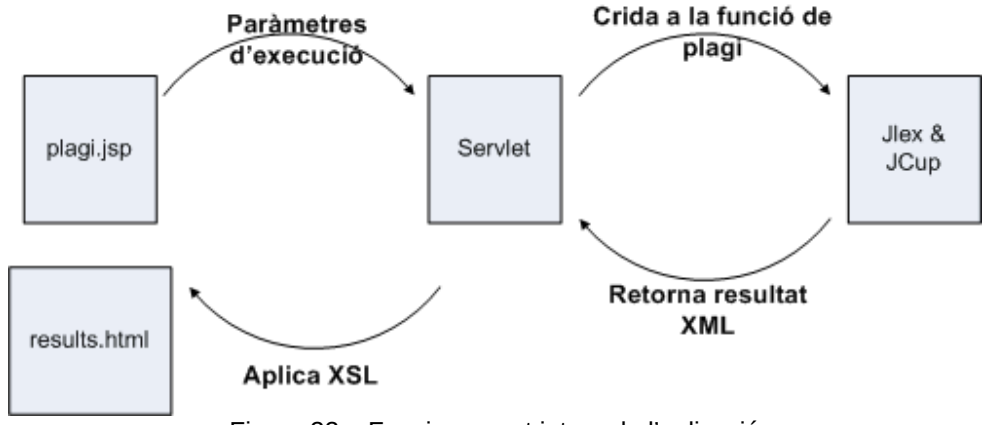


Figura 22 – Funcionament intern de l'aplicació

Com podem veure la idea principal és proporcionar els paràmetres d'entrada a la funció de plagi perquè ens retorni un XML amb els resultats de la comparació.

Un cop explicat el funcionament global, entrem en el detall de la funció de plagi que es troba al arxiu *plagi.cup* amb el nom de *plagiarism*. Aquesta comprova que la crida sigui correcta, parceja els programes i registra el seu resultat que pot prendre els valors de *OK*, *FileNotFound* i *Fail*. Una vegada processats tots els programes els tenim a la variable de tipus *TProgram*.

Per realitzar la comparació dels programes és crida al mètode *compareProgram*s de la classe *TProgram*, aquest s'encarrega de fer les comparacions i aplicar el algorisme *Greedy String Tiling*, el pseudocodi de la funció és el següent:

A continuació es mostra el codi de la funció de comparació:

```
compareProgrames() {
        for(int i=0; i<tPrograms.size();i++) {
            TSimbol compare = tPrograms.getProgram(i);
            for(int j=i+1;j<tPrograms.size();j++) {
                TSimbol pivot = this.getProgram(j);
                Compare result = greedyStringTiling(compare,pivot);
             }
        }
}
```

A partir del pseudocodi es pot deduir la fórmula per calcular el nombre de comparacions que es realitzaran donats n programes.

Per exemple si s'han parcejat 8 programes, el nombre de comparacions que es realitzarà serà de 7 + 6 + 5 + ... + 1 = 28 comparacions. És a dir que donats n programes el nombre de comparacions és el sumatori de (n - 1). Si agrupem aquest sumatori obtenir la següent fórmula:

F(n) = n*(n-1)/2

Si n=8. F(8) = 8*(8-1)/2 = 28 comparacions.

Per cada comparació, si el seu percentatge de similitud està per sobre el tall que ha establert l'usuari es guarda la comparació a la variable de tipus ResultsParser, de manera que únicament s'escriuen les comparacions que han donat per sobre el tall especificat per l'usuari.

8 Proves de l'eina automàtica de plagis

8.1. Jocs de proves

8.1.1. Proves inicials

El primer joc de proves que s'utilitza per avaluar l'eina és exactament el mateix que s'ha utilitzat en les aplicacions anteriors. Com s'ha comentat anteriorment cada prova conté totes les entregues parcials i finals que ha realitzat un alumne, aquest fet implica que el nombre d'arxius sigui molt elevat.

Les proves es fan amb Minimum Match Length: 12

Fonaments de Programació	Temps d'execució	Prog. Processats	Prog. Error
Pac1	3 minuts 1 segon	1493	136
Pac2	5 minuts 23 segons	1003	171
Pac3	4 minuts 66 segons	989	91
Pac4	22 minuts 15 segons	1288	173
Pràctica	60 minuts 5 segons	1535	162

Degut al gran nombre de programes el temps d'execució són molt elevats, en el cas de la Pac4 i la Pràctica s'arriben a temps que sobrepassen l'espera màxima d'un usuari.

A continuació mostrem els gràfic que han generat les execucions:

Pac1:

Similarity

%	Programs Matching	
90% - 100%	159464	**********
80% - 90%:	126635	**********
70% - 80%	64294	*****
60% - 70%	75625	*******
50% - 60%	62441	*****
40% - 50%	54854	*****
30% - 40%	20044	*
20% - 30%	88	-
10% - 20%	0	-
0% - 10%	404691	**********

Figura 23 – Execució de la Pac1

Pac2:

Similarity

%	Programs Matching	
90% - 100%	69508	**********
80% - 90%:	77277	***********
70% - 80%	34697	******
60% - 70%	27583	*****
50% - 60%	19594	***
40% - 50%	25038	****
30% - 40%	12496	**
20% - 30%	15648	***
10% - 20%	4830	-
0% - 10%	164104	**********

Figura 24 – Execució de la Pac2

Pac3:

Similarity

%	Programs Matching	
90% - 100%	66924	**********
80% - 90%:	52200	**********
70% - 80%	36049	*******
60% - 70%	29777	*****
50% - 60%	31511	******
40% - 50%	10921	**
30% - 40%	14114	**
20% - 30%	8705	*
10% - 20%	3168	-
0% - 10%	203121	**********

Figura 25 – Execució de la Pac3

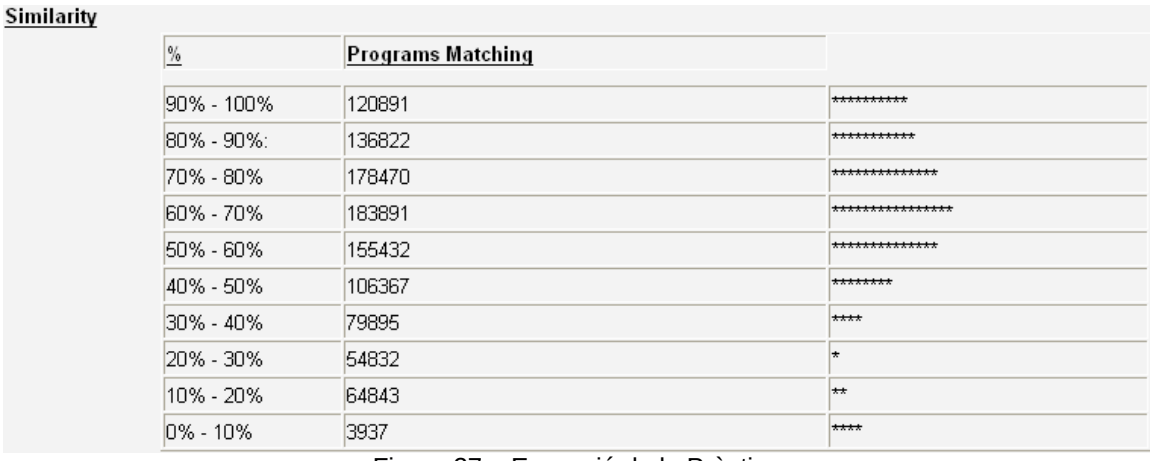
Pac4:

Similarity

%	Programs Matching	
90% - 100%	60101	******
80% - 90%:	64243	*******
70% - 80%	58985	******
60% - 70%	62552	*******
50% - 60%	92024	**********
40% - 50%	80076	*********
30% - 40%	71231	********
20% - 30%	66685	*******
10% - 20%	95229	**********
0% - 10%	147090	**********

Figura 26 – Execució de la Pac4

Pràctica:



Similarity

%	Programs Matching	
90% - 100%	120891	**********
80% - 90%:	136822	**********
70% - 80%	178470	**********
60% - 70%	183891	**********
50% - 60%	155432	**********
40% - 50%	106367	********
30% - 40%	79895	****
20% - 30%	54832	*
10% - 20%	64843	**
0% - 10%	3937	****

Figura 27 – Execució de la Pràctica

En els gràfics podem observar el nombre de comparacions que es troben en un percentatge determinat. D'aquesta forma podem saber en quin rang s'han donat el major nombre de comparacions.

Com podem observar amb aquests jocs de proves el nombre de comparacions que s'ha de mostrar és molt elevat, per exemple en l'execució de la pràctica suposant un tall del 80% es tindrien que mostrar 257.713 resultats. És per això que en el següent apartat es tracten els jocs de proves amb la intenció de millorar-los.

8.1.2. Proves última entrega

Tenint en compte que únicament s'avalua la última entrega que realitza un alumne, per millorar els jocs de proves es passa un procés per obtenir únicament l'ultima entrega realitzada. Com es pot observar el nombre de fitxers a tractar disminueix considerablement.

Fonaments de Programació	Temps d'execució	Prog. Processats	Prog. Error
Pac1	34 segons	449	54
Pac2	1 minut 1 segon	437	74
Pac3	1 minut 1 segon	421	41
Pac4	2 minuts 11 segons	381	50
Pràctica	6 minuts 24 segons	390	53

El temps d'execució s'han reduït considerablement, els quals es consideren temps acceptables d'espera en donar un resultat a l'usuari.

A continuació mostrem els gràfics que han generat les execucions:

Pac1:

Similarity

%	Programs Matching	
90% - 100%	18771	**********
80% - 90%:	14049	********
70% - 80%	5076	****
60% - 70%	7900	******
50% - 60%	5094	****
40% - 50%	5230	****
30% - 40%	1778	*
20% - 30%	11	-
10% - 20%	0	-
0% - 10%	45831	**********

Figura 28 – Execució de la Pac1

Pac2:

Similarity

%	Programs Matching	
90% - 100%	13178	**********
80% - 90%:	13037	**********
70% - 80%	5186	*****
60% - 70%	5370	*****
50% - 60%	3279	***
40% - 50%	4745	*****
30% - 40%	2546	**
20% - 30%	3428	***
10% - 20%	889	-
0% - 10%	32187	**********

Figura 29 – Execució de la Pac2

Pac3:

Similarity

%	Programs Matching	
90% - 100%	15909	**********
80% - 90%:	11138	********
70% - 80%	7041	*******
60% - 70%	5995	******
50% - 60%	5890	******
40% - 50%	2147	**
30% - 40%	2548	**
20% - 30%	1687	*
10% - 20%	495	-
0% - 10%	28556	**********

Figura 30 – Execució de la Pac3

Pac4:

Similarity

%	Programs Matching	
90% - 100%	5508	*******
80% - 90%:	5272	******
70% - 80%	4929	******
60% - 70%	4802	******
50% - 60%	7207	*********
40% - 50%	5912	*******
30% - 40%	5840	*******
20% - 30%	4622	******
10% - 20%	11037	**********
0% - 10%	13506	**********

Figura 31 – Execució de la Pac4

Pràctica:

Similarity

%	Programs Matching	
90% - 100%	8225	**********
80% - 90%:	9684	**********
70% - 80%	12112	**********
60% - 70%	13284	**********
50% - 60%	11490	**********
40% - 50%	6603	********
30% - 40%	3850	****
20% - 30%	1568	*
10% - 20%	2400	**
0% - 10%	3937	****

Figura 32 – Execució de la Pràctica

Amb el procés realitzat als jocs de proves s'ha reduït el nombre d'arxius, això implica que el temps d'execució sigui menor així com el nombre de resultats que s'han de mostrar.

Ara per a l'execució de la Pràctica, suposant el mateix tall del 80% es tindrien que mostrar a l'usuari 17909 comparacions. Aquest nombre tot i ser molt menor al anterior continua sent molt elevat, aquest és conseqüència del elevat nombre d'estudiants en una assignatura de la UOC.

8.2. Resultats obtinguts

A partir de les proves realitzades s'ha realitzat algun ajust a l'eina per tal de millorar la seva interacció amb l'usuari.

Com s'ha observat el nombre de comparacions que s'han de realitzar per el volum d'entregues que es donen en una assignatura de la UOC és molt elevat.

Per limitar el nombre de resultats que es mostren a l'usuari aquest pot indicar el percentatge mínim que s'ha de donar per mostrar els detalls de la comparació. Tot i això encara que l'usuari especifiqui un percentatge mínim del 90%, en les proves realitzades el nombre de resultats continua sent molt elevat.

Per reduir el temps de càlcul i limitar nombre de fitxers que s'escriuen per mostrar els resultats, s'ha introduït un límit que s'especifica en el fitxer de configuracions.

En les proves realitzades aquest límit és de dos cents resultats, és a dir que únicament mostrarem els primers dos cents resultats majors al rang especificat per l'usuari.

Aquesta mesura ve donada principalment per dos motius:

- Limitar l'escriptura de resultats: d'aquesta manera controlem que un usuari faci una execució que pogués malmetre el funcionament del servidor.
- Reduir el temps de càlcul.

Aquest límit estarà en funció de les necessitats i potència del servidor on s'executa l'aplicació.

A continuació es mostra el log que ha generat l'execució de la Pac1 que conté la última entrega de cada alumne:

```
Inici: 20:15:33
Resultats:

      Parsing Total de Programes: 499
      Programes OK:               445
      Programes Fail:             54
      Programes FileNotFound:     0

Programes comparats: 456
Sumatori: 103740
Fi Compara
Fi: 20:16:07
```

Com podem observar el nombre programes que s'han pogut processar correctament és de 445, però el nombre de programes que es comparen és de 456. Això és degut a que alguns dels programes que han fallat si que s'han pogut generar Tokens fins al punt on s'ha produït l'error.

Es per això que si sumem la columna del gràfic Programs Matching el seu resultat dona 103740 comparacions.

En funció del criteri de l'usuari aquest podria preferir que no és comparessin les pràctiques que no s'han pogut processar completament, d'aquesta forma també es redueixen el nombre de comparacions que s'han de realitzar però també és podria donar el cas que es passes per alt algun cas de plagi.

9. Instal·lació del software

En aquest capítol s'explica la instal·lació de l'eina sobre una màquina GNU/Linux i es mostra un exemple d'execució.

9.1. Guia d'instal·lació

Els requeriments l'eina són els següents:

- Tomcat[15] >= 4
- Compilador javac[16] >= 1.4.2
- Xalan-Java[17]
- JLex[18]
- JCup[19]
- Llibreria cos.jar[20]

L'aplicació es troba dintre la carpeta plagi, aquesta segueix l'organització per a que es pugui copiar dintre el directori *webapps* del servidor Tomcat i accedir amb el navegador per exemple amb la següent direcció: http://localhost:8080/plagi

La carpeta plagi està organitzada amb els següents directoris:

- Directori arrel: conté la pàgina principal de l'aplicació *plagi.jsp*, el fitxer XSL *plagi.xsl* per aplicar la conversió al resultat en XML, la DTD *plagi.dtd* per validar el XML generat, l'arxiu README.txt amb les explicacions per instal·lar l'aplicació i l'arxiu *help.html* que fa servir l'aplicació per mostrar l'ajuda a l'usuari.
- fonts: en aquest directori es troben els fitxers *.java, *JLex, JCup*, el directori amb les classes JLex, el directori amb les classes JCup i la llibreria cos.jar.
  Per compilar la pràctica s'ha creat l'script *compile.sh*, que compila i copia les classes en els directoris corresponents.
- WEB-INF: aquest directori conté l'arxiu *web.xml* que configura l'aplicació, en ell es declara el servlet, la pàgina d'inici, etc.
  L'arxiu *config.properties* conté les variables de configuració que fa servir l'aplicació.
  Al directori *classes* es troben els fitxers compilats (servlet, etc) de les classes utilitzades per l'aplicació. Dintre es troben les classes proporcionades per JLex i JCup en els seus corresponents directoris, el directori plagiarism que conté les classes compilades que em creat i el servlet compilat. El directori Filestmp es utilitzat per l'aplicació per realitzar la comparació.

[15] http://tomcat.apache.org/
[16] http://www.java.com/es/
[17] http://xml.apache.org/xalan-j/
[18] http://www.cs.princeton.edu/~appel/modern/java/JLex/
[19] http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html
[20] http://servlets.com/cos/

Al directori *lib* es troben les biblioteques de classes addicionals comprimides amb *jar* que fa servir l'aplicació. En el nostre cas *cos.jar* que serveix per pujar els arxius al servidor.

- results: aquest directori conté els directoris amb les execucions. Un directori per execució, cada directori conté el fitxer results.xml, results.html, i els fitxers *.html amb detall de cada comparació.
- img: conté les imatges que fan servir les pantalles de l'aplicació.

Els passos que s'han de seguir per instal·lar l'aplicació des de zero en un sistema GNU/Linux són:

1. Instal·lar i configurar les eines javac, Tomcat i Xalan-Java.
2. Descomprimir l'arxiu zip i copiar la carpeta dintre el directori webapps.
   shell> unzip plagi.zip
   shell> mv –r plagi/ <webappsDir>
3. Donar permisos d'escriptura al usuari que executa el servidor Tomcat als directoris results i Filestmp.
4. Compilar l'aplicació (opcional): tot i que aquest pas és opcional ja que l'aplicació està ja compilada, es recomana compilar-la per comprovar la correcte configuració del servidor on s'executa l'aplicació.
   shell> ./compile.sh
5. Configurar les variables del fitxer config.properties, renombrar la ruta on s'ha copiat la carpeta plagi.
   dFilesTmp=**/home/plagis**/plagi/WEB-INF/classes/Filestmp/
6. Execució mode comandes (opcional): en aquest pas s'executa l'aplicació en mode comandes i ens serveix per comprovar els passos anteriors. Per executar l'aplicació en mode comandes copiem uns fitxers *.c al directori Filestmp. A continuació executem la següent comanda.
   shell> java plagiarism.parser provaExecucio
   Aquesta execució ens mostrarà el log de l'aplicació per pantalla i el resultat el deixarà dintre la carpeta results al directori provaExecucio.
7. Configurar els arxius jsp i xsl, perquè apuntin al servidor correcte.
8. Execució via web: obrir el navegador i introduir la següent direcció.
   http://localhost:8080/plagi

9.2. Exemple d'execució

Els passos a seguir per executar l'aplicació són els següents:

1. Comprimir els fitxers *.c en un fitxer *.zip
2. Obrir un navegador i introduir la direcció HTTP per accedir a l'eina de plagis.

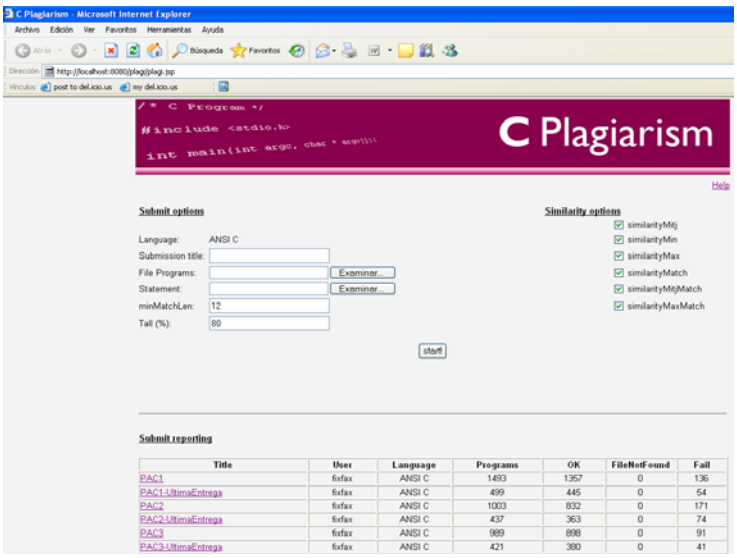


Figura 33 – Pàgina inicial

3. Introduir com a mínim el títol de l'execució i seleccionar el fitxer *.zip que conté els fitxers *.c que es volen comparar.
4. Una vegada introduïts els paràmetres d'execució clissar start!.
5. Esperar a que es mostri la pantalla amb els resultats de l'execució.

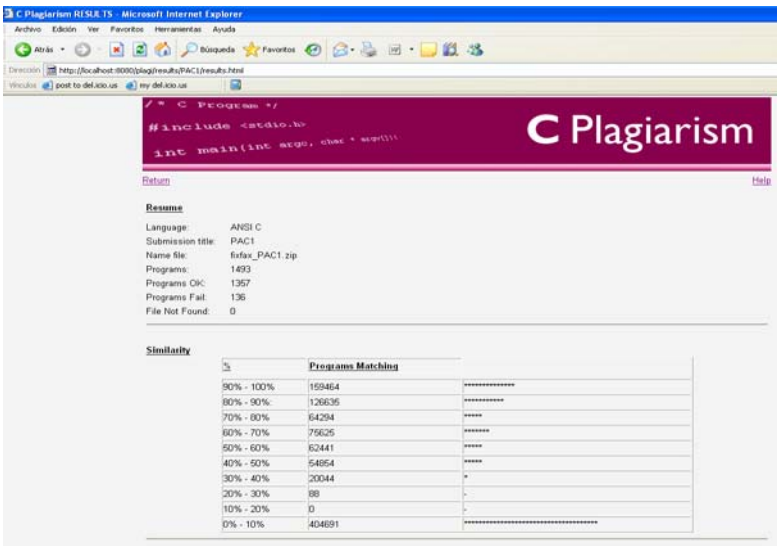


Figura 34 – Pàgina resultats

6. A partir de la pantalla de resultats ja podem navegar per obtenir el detall d'una comparació.

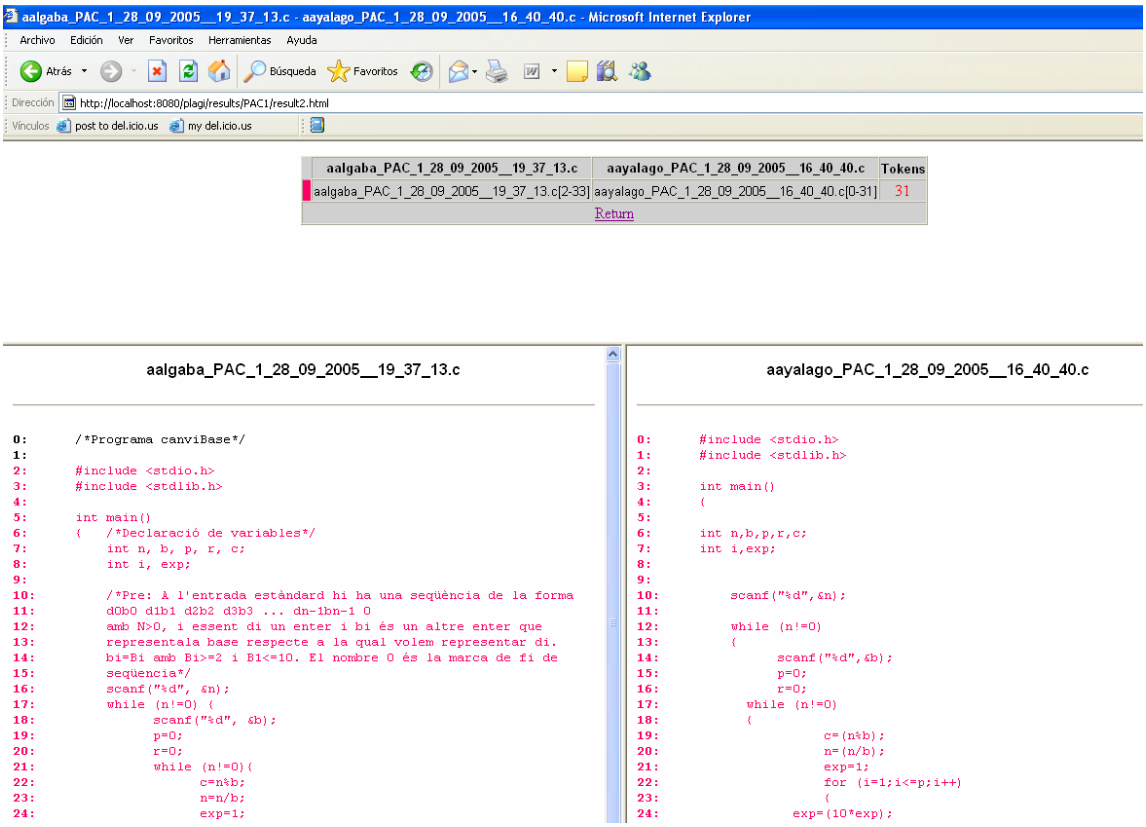


Figura 35 – Detall d'una comparació

10. Conclusions

Arribat a la fi del projecte, es fa una recol·lecta de les principals conclusions que s'han extret i una valoració dels objectius inicials establerts.

10.1. Abast de l'eina de plagis

Durant el desenvolupament del projecte s'ha anat prenent consciència de la complexitat que té la creació d'una eina de plagis per a un àmbit universitari. Ja que a la complexitat general del projecte, crear un compilador per fer la traducció a tokens, implementar la funció de plagi, etc, s'afegeix el volum de dades que s'ha de tractar en conseqüència del nombre d'estudiants que té la UOC, així fer la representació de les dades.

L'objectiu principal del projecte era obtenir una eina per detectar el plagi en codi font escrit en C. Després d'aplicar les tècniques i avaluar l'eina amb un nombre molt elevat de pràctiques, es considera que encara no està preparada per declinar tota la responsabilitat a l'eina, sinó que proporciona una ajuda al professor per realitzar aquesta tasca.

Aquesta necessita tenir un major temps de vida per tenir més experiència, i també perquè els usuaris valorin la seva efectivitat i afinar-la al tipus de pràctiques que es corregeixen.

En funció de l'efectivitat, desenvolupament i millores posteriors que es puguin fer a l'eina és tindran més en compte els resultats proporcionats per aquesta, l'objectiu més ambiciós seria poder declinar a l'eina la responsabilitat de dictaminar els casos de plagi.

També cal comentar que a partir de la base creada, aquesta es pot adaptar fàcilment a altres llenguatges de programació. Per exemple un llenguatge utilitzat en un gran nombre de pràctiques és Java.

10.2. Valoració

Els objectius principals del projecte eren els següents:

1. Introduir al lector en el món de la detecció del plagi, en concret en la comparació de fitxers de codi font.
2. Instal·lar, avaluar i proporcionar un catàleg de les eines ja existents.
3. Estudiar l'algorisme GST i mostrar el seu funcionament.
4. Generar una eina automàtica de detecció de copies que pugui ser utilitzada per l'equip docent de la UOC.
5. Finalment es comparà aquesta nova eina amb les existents i s'extrauran les conclusions pertinents.

Explicació de com s'han aconseguit els punts anteriors:

1. S'ha realitzat una introducció de que és el plagi i una explicació del seu cicle de vida.
2. Utilització de les eines de plagi per elaborar un resum amb les seves funcionalitats i particularitats, amb aquest catàleg l'usuari també pot accedir a les eines per comprovar el seu funcionament.
3. Explicació del codi de l'algorisme i exemple de funcionament.
4. S'adjunta al projecte tot el codi de l'aplicació i s'ha elaborat un manual d'instal·lació i utilització.
5. S'han processat els mateixos jocs de proves a totes les eines i s'ha comprovat que els temps d'execució són acceptables.

11. Glossari

- **Desenvolupament en W**: simplificació del mètode espiral. Obtenció d'un prototip al primer cicle, a partir d'aquest es realitza un segon anàlisi per implementar el producte final.
- **Eina automàtica de detecció de còpies**: sistema automàtic per analitzar documents per detectar plagis parcials o totals.
- **EPOO**: Acrònim de Enginyeria del Programari Orientat a Objecte.
- **Greedy String Tiling**: algoritme de detecció de plagis en parells de documents.
- **GST**: Acrònim de *Greedy String Tiling*.
- **HTML**: acrònim d'*Hypertext Markup Language*.
- **Java**: Tecnologia desenvolupada per *Sun Microsystems* per desenvolupar aplicacions independents de la plataforma on s'executa.
- **JCup**: sistema realitzat en Java, que s'utilitza per generar analitzadors sintàctics *LALR*.
- **JLex**: Generador automàtic d'analitzadors lèxics, realitzat en Java, el codi de l'explorador generat també és en Java.
- **JPlag**: Eina de detecció de plagis utilitzada principalment per a la detecció de codi font, encara que també suporta el llenguatge natural.
- **LALR**: Acrònim de *Look-Ahead LR*.
- **Llenguatge C**: Llenguatge de programació creat al 1969 per Ken Thompson i Dennis M. Ritchie als laboratoris Bell, prenen com a referència els llenguatges BCPL i B.
- **Minimum Match Length**: longitud mínima de coincidència de tokens entre dos fitxers de codi font per tal que siguin considerats com a plagi.
- **MOSS**: Acrònim de *Measure Of Software Similarity*. Eina de detecció de plagis utilitzada principalment per a la detecció de codi font.
- **Match**: coincidència d'una cadena text entre dos documents.
- **Parser**: Programa informàtic que efectua un *parcing* sobre una seqüència d'entrada.
- **Parsing**: Procés informàtic que analitza una seqüència d'entrada per determinar l'estructura gramatical respecte una determinada gramàtica.
- **Plagi**: còpia sense autorització de l'autor o de qui posseeix els drets, i presentar-lo com una obra pròpia.
- **Percentatge de similitud**: mesura per quantificar la igualtat en un parell de documents.
- **PFC**: Acrònim de Projecte Final de Carrera.
- **SIM**: Eina de detecció de plagis utilitzada principalment per a la detecció de codi font.
- **Tile**: conjunt de tokens iguals superior a *MML*.
- **Tokens**: en aquest context, part mínima del fitxer de codi font que és susceptible de ser comparada amb un altre fitxer de codi font.
- **UOC**: Acrònim de Universitat Oberta de Catalunya.
- **WINNOWING**: algoritme utilitzat per a la detecció del plagi.

- **XML**: Acrònim *d'eXtensible Markup Language*.
- **YAP**: Acrònim de *Yet Antoher Plague*. Eina de detecció de plagis utilitzada principalment per a la detecció de codi font.

12. Bibliografia

Ref. 1. "**Apunts Interacció humana amb els ordinadors**". 3 de desembre del 2004.

Ref. 2. Lutz Prechelt, Guido Malpohl, Michael Phlippsen: "**Finding Plagiarisms among a Set of Programs with JPlag**". Document electrònic, 30 de març del 2000.
http://www2.informatik.uni-erlangen.de/Forschung/Publikationen/download/jplag.pdf?language=de

Ref. 3. Michael J Wise: "**Running Karp-Rabin Matching and Greedy String Tiling**". Document electrònic, 1 de març del 1993.
http://www.it.usyd.edu.au/research/tr/tr463.pdf

Ref. 4. Xin Chen, Brent Francia, Ming Li, Brian Mckinnon, Amit Seker: "**Shared Information and Program Plagiarism Detection**". Document electrònic, 13 de desembre 2003.
http://monod.uwaterloo.ca/papers/04sid.pdf

Ref. 5. Saul Schleimer, Daniel S. Wilkerson, Alex Aiken: "**Winnowing: Local Algorithms for Document Fingerprinting**". Document electrònic.
http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf

Ref. 6. Dick Grune, Matty Huntjens: "**Detecting copied submissions in computer science workshops**". Document electrònic, 1 de novembre del 1989.
ftp://ftp.cs.vu.nl/pub/dick/similarity_tester/Paper.ps

Ref. 7. Elliot Berk: "**JLex: A lexical analyzer generator for Java**". Document electrònic, 5 de maig de 1997.
http://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html

Ref. 8. Scott E. Hudson: "**CUP User's Manual**". Document electrònic, 1 de juliol del 1999.
http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html

Ref. 9. "**Java API Specification**". Pàgina web.
http://www.ccd.uab.es/~sergi/manuals/javaAPI/docs/api/index.html

Ref. 10. "**Informació XML**". Pàgina web.
http://www.devx.com/xml/Door/7050