Università degli Studi di Perugia

DIPARTIMENTO DI MATEMATICA ED INFORMATICA

Corso di Laurea in Informatica

Tesi di laurea

IMPLEMENTAZIONE DI UN TOOL PER IL SUPPORTO ALLA VALUTAZIONE AUTOMATICA DI PROGETTI SU GITHUB

Laureando:
Matteo Azzarelli
Matricola 244999

Relatore:
Prof. Francesco Santini

Anno Accademico 2017-2018

Ringrazio il Prof. Francesco Santini
per i preziosi insegnamenti che mi ha trasmesso.

Un ringraziamento particolare ai miei genitori
e a Luisa per il grande supporto datomi
in questi anni che mi ha permesso
di raggiungere questo importante traguardo.

Indice

Elenco delle figure

Introduzione

Questa applicazione nasce dall'esigenza di trovare le similitudini tra una moltitudine di progetti di esami scritti in linguaggio `C`.

Tenere sotto controllo centinaia di progetti, ricordandosi ogni singolo codice è umanamente impossibile, per questo motivo il seguente programma è un'eccellete supporto alla valutazione. Infatti il docente non ha più l'onere di memorizzare ogni codice alla ricerca di illeciti, piuttosto potrà concentrarsi sulla valutazione delle effettive capacità degli alunni. Lo scopo quindi è di analizzare i compiti degli studenti incrociando gli esami dello stesso appello e se necessario anche degli appelli precedenti, così da ridurre al minimo la probabilità di sfuggire al controllo.

Ci auguriamo che gli studenti prendendo coscienza dell'effettiva esistenza di un software di controllo funzionante siano dissuasi dal perpetrare le azioni di copiatura. Questa teoria è anche supportata da tutti gli sviluppatori di applicazioni di plagiarism detection.

In questo progetto vedremo cosa è GitHub (1.1) e GitHub Classroom (1.1.2) con annessa introduzione al software Git (1.1.1). Dopodiché andremo alla scoperta dei maggiori programmi di plagiarism detection (2) e in fine vedremo più in dettaglio come è stata realizzata l'applicazione (3) che permette di scaricare da GitHub i progetti di esame degli studenti e in seguito di analizzare li stessi tramite il supporto di MOSS (3.2).

1 | GitHub Classroom

1.1 GitHub

GitHub è un servizio di hosting per il controllo delle versioni basato su *Git* (1.1.1). Esso, oltre al controllo delle versioni, fornisce anche altre funzionalità di collaborazione come il bug tracking, gestione delle attività e wiki per ogni progetto.

GitHub offre la possibilità di creare repositories sia private che pubbliche, quest'ultime spesso sono utilizzate per condividere progetti open-source, molto importante per la comunità scientifica.

Nato nel 2008 questo progetto prende subito piede, infatti in nemmeno un anno raggiunge 100.000 utenti iscritti. Dopo 10 anni dalla sua nascita arriva a 22 milioni di iscrizioni diventando uno standard e un requisito necessario per tutti i programmatori.

Un importantissima iniziativa lanciata da GitHub è il nuovo programma **GitHub Student Developer Pack** che concede gratuitamente agli studenti un insieme dei tools e servizi più popolari come ad esempio *awsEducate*, *bitnami* o ancora *Microsoft Azure* e ovviamente *GitHub* offre repositories private illimitate. GitHub offre agli insegnanti il tool chiamato **GitHub Classroom** (1.1.2), che si propone come uno strumento per aiutare gli insegnanti ad educare le nuove generazioni di sviluppatori ad utilizzare gli strumenti richiesti dalle aziende e a padroneggiare il linguaggio necessario per immettersi nel mondo del lavoro.

1.1.1 Git

Una breve digressione sul *controllo di versione distribuito* ***Git*** [Chacon and Straub, 2018].

Esso tiene traccia dei cambiamenti dei file e coordina il lavoro su questi file tra un team di più persone. È utilizzato prevalentemente per la gestione di codice sorgente di progetti di software, ma esso può essere utilizzato per tenere traccia dei cambiamenti in qualsiasi set di files.

Pensato per essere un sistema di controllo distribuito esso è stato progettato per mantenere l'integrità dei files, di garantire un'alta velocità e anche di gestire dei flussi di lavoro non lineari.

Questo magnifico software è stato creato da *Linus Torvalds* nel 2005 insieme ad altri suoi colleghi per facilitare lo sviluppo del kernel di Linux.

Git si distingue da tutti gli altri software di versione per il modo in cui immagazzina i dati. A differenza dei precedenti VCS che salvano i dati come una lista delle modifiche (Figura 1.1), Git salva i dati come delle istantanee (Figura 1.2).

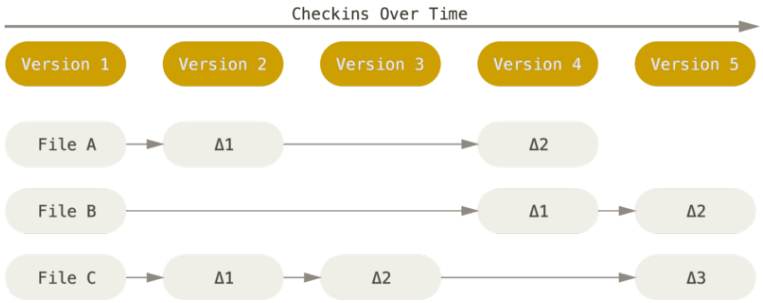

Figura 1.1: Gli altri VCS tendono ad immagazzinare i dati come cambiamenti alla versione base di ogni file.

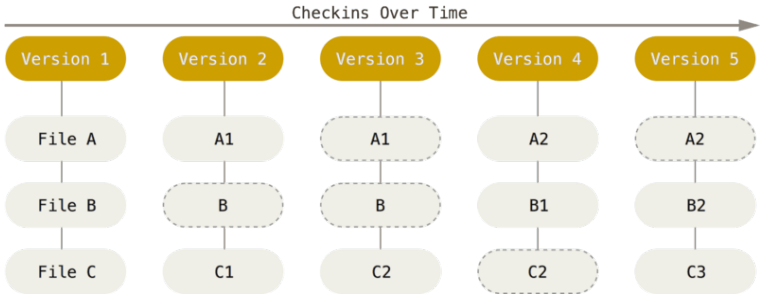

Figura 1.2: Git immagazzina i dati come snapshot del progetto nel tempo.

Questa peculiarità rende Git simile ad un mini *filesystem* con tutti i benefici di un gestore delle versioni. Questo modo di pensare i dati rende possibile la creazione di differenti Branch paralleli, in seguito verranno analizzati.

Ora esploriamo brevemente i controlli principali di Git:

- Init
- Clone
- Commit
- Fretch
- Push
- Pull

Per gli altri comandi si rimanda alla guida ufficiale ProGit [Chacon and Straub, 2018].

Init

Se hai una directory di un progetto e vuoi iniziare il controllo di versione su di essa con Git dovrai utilizzare il comando:

```
$    git init
```

Questo comando crea una nuova sottodirectory chiamata .git, la quale contiene tutti i file necessari per la repository. A questo punto ancora non è tenuta traccia di alcun file. Se vuoi iniziare il controllo di versione su file già preesistenti dovrai aggiungere i file alla repository e poi fare un commit iniziale, ad esempio aggiungiamo tutti i file `.c` e un file di licenza e facciamo il commit con un messaggio:

```
$    git add *.c
$    git add LICENSE
$    git commit -m 'progetto versione iniziale'
```

A questo punto abbiamo una Git repository con i file tracciati e con un commit iniziale.

Clone

Molto spesso ci troviamo a lavorare con progetti già avviati, quindi in questo caso dovremmo utilizzare il comando `git clone <url>` per, appunto, clonare la repository desiderata. Esempio:

```
$    git clone https://github.com/MatteoAzzarelli/GitHub
ClassroomDownloader.git
```

In questo modo verrà creata una directory chiamata ClassroomDownloader, già inizializzata con la cartella .git, dove verranno scaricati tutti i file del progetto pronti per essere utilizzati.

Commit

Forse è il comando più utilizzato, in quanto permette di rendere effettive le modifiche ai file. Aggiungendo il flag `-m` abbiamo la possibilità di mettere un commento al commit, così da poter capire e far capire anche ai collaboratori quale modifiche sono state apportate. Esempio:

```
$    git commit -m 'Fix al Bug#'
```

Fetch

Per prendere i dati dal progetto remoto, ad esempio per aggiornare i file con le modifiche apportate da un collaboratore, è possibile utilizzare il seguente comando:

```
$    git fetch <remote>
```

dove <remote> è il nome del server remoto. si possono visualizzare i server remoti tramite il comando `git remote` , il server di default è chiamato `origin`.

Push

Nel momento in cui vuoi condividere con gli altri collaboratori le tue modifiche, dovrai caricarle nel server i tuoi file con il comando se `git push <remote> <branch>`. Esempio:

```
$    git push origin master
```

Il branch di default è chiamato master. Un branch è un ramo del progetto, questi spesso vengono utilizzati per aggiungere delle features al progetto, senza influire sulla versione base se la modifica non dovesse andare a buon fine. I rami possono terminare senza essere integrati nella versione base come si vede nel branch *Feature 2* in Figura 1.3, oppure possono essere riuniti ad un altro branch come ad esempio *Feature 1* o *Develop* in Figura 1.3.

Figura 1.3: Esempio di un workflow a branch

Pull

Il comando `git pull` generalmente recupera i dati dal server da cui originariamente sono stati clonati e tenta automaticamente di unirlo al codice su cui stai attualmente lavorando. Se non dovesse riuscire ad unire i file chiederà all'utente di risolvere i conflitti generati.

1.1.2 GitHub Classroom

GiHub Classroom è lo strumento messo a disposizione da GitHub per gli insegnanti. Gli step fondamentali per iniziare ad usarlo sono:

- **Nuova organizzazione** Creare un'organizzazione dal pannello personale di GitHub o se già esistente sfruttare quest'ultima.
- **Nuova Classroom** Il passo successivo è creare una nuova Classroom. Si dovrà selezionare l'organizzazione desiderata per poi inserire il nome della classe. C'è la possibilità di invitare altri insegnanti alla classe appena creata o dei membri.
- **Nuovo Compito** A questo punto si può procedere alla creazione di un nuovo *assignment* ovvero un nuovo compito. È possibile fare scegliere se assegnare compiti di gruppo o individuali. Ogni compito ha un titolo, può essere reso pubblico o privato, si può anche aggiungere una repository per dare un insieme di file di base.

2 | Lavori correlati

Gli autori di [Parker and O. Hamblen, 1989] definiscono il plagio di software come "un programma che è stato prodotto da un altro e riproposto con un esiguo numero di trasformazioni di routine".

Le trasformazioni che possono aver luogo possono variare da quelle più semplici (come cambiare i commenti nel codice oppure cambiare i nomi delle variabili) a quelle più complesse (rimpiazzare strutture di controllo con altre equivalenti, ad esempio sostituire un ciclo "for" con un "while"). Questo può essere visualizzato usando i sei livelli rappresentati nello spettro del plagio (figura 2.1) realizzato da [Faidhi and Robinson, 1987].

Una proprietà importante di qualsiasi sistema è la corretta individuazione di codice sospetto. C'è una netta differenza tra similarità di codice nata casualmente e dovuta al plagio. Questo è molto simile a individuare parole o frasi "inusuali" nel testo scritto.

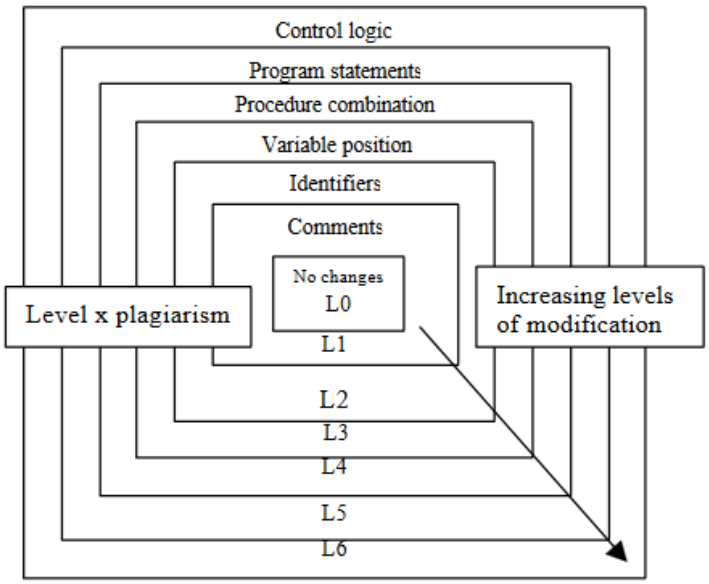

Figura 2.1: Raffigura i vari livelli di plagio all'interno di un programma

Una fonte molto utile per la detenzione del plagio nei software è l'aticolo di [Whale, 1990] che ha fatto una lista dei potenziali metodi o stratagemmi per camuffare il plagio, inclusi i seguenti:

- Cambiare i commenti,
- Cambiare il tipo di dato (es. da float a double),
- Cambiare il nome delle variabili,
- Aggiungere dichiarazioni o variabili ridondanti,
- Modificare le strutture degli statements di selezione (es. modificare la struttura di un if),
- Combinare porzioni di codice copiato ed originale.

Whale prosegue discutendo la valutazione dei sistemi di rilevamento dei plagi e di rilevamento della similitudine, tra cui anche il suo stesso tool chiamato Plague (2.3).

Ora presentiamo una breve rassegna sui migliori software di *plagiarism detection*.

2.1 SIM (Software Similarity Tester)

SIM è stato svilupppato da Dick [Grune] a Vrije Universiteit. Questo software è distribuito gratuitamente sotto forma di sorgente in linguaggio `C` e testa la similarità tra programmi misurando l'affinità lessicale di testi scritti in *C*, *Java* , *Pascal*, *Modula-2*, *Miranda*, *Lisp*, *8086 assembler code* e il linguaggio naturale.

Dick Grune afferma che SIM è in grado di scovare il plagio nei progetti software e di trovare frammenti di codice parzialmente duplicati all'interno di quest'ultimi anche se di grandi dimensioni.

L'output di SIM può essere processato utilizzando script da shell per produrre istogrammi o una lista di sottomissioni sospette.

2.2 Siff

L'applicazione originale di questo strumento era trovare file simili in un sistema di grandi dimensioni ed il suo nome originario era *Sif* La tecnica era stata utilizzata in concomitanza con altri metodi per misurare la somiglianza nei file bytecode Java. Il programma è stato poi rinominato "**Siff**".

Siff era stato inizialmente progettato per confrontare un gran numero di file di testo per trovare affinità tra di loro. Siff è un tool di UNIX che può confrontare due file di testo per similarità, ma assumendo 1 secondo per ogni confronto, tutti confronti a coppie tra 5000 file richiederebbe più di 12

milioni di confronti e la CPU impiegherebbe all'incirca 5 mesi per completare il lavoro.

Il nuovo algoritmo di Siff cerca di ridurre questo tempo. I file vengono rappresentati in una forma compatta, la "approximate fingerprint". Questa fingerprint può essere confrontata velocemente, ma concede differenze nei file, solo una somiglianza del 25%). Se i due file hanno 5-10 fingerprints in comune, vengono classificati come "simili", dipenderà dalla dimensione del file. Il vantaggio di questo metodo è che il fingerprint tra due file simili avrà una grande intersezione e due file non in relazione avranno invece una piccola intersezione molto probabilmente. Di solito la probabilità di trovare la stessa stringa di 50 byte in due file non in relazione è molto scarsa ma il metodo è suscettibile a "cattive" fingerprints. Per esempio, quando si scrive un testo di piccole dimensioni è facile trovare molte somiglianze con altri file. Questo può portare a identificare simili due file non in relazione. Questo può avvenire estraendo soltanto del testo dai documenti.

I vari usi di Siff possono essere:

- Confrontare differenti revisioni del codice di un programma per il plagio.
- Trovare copie duplicate di file in un grande archivio di dati.
- Trovare file simili in Internet così da ridurre la ricerca in cartelle dove ci potrebbero essere file simili come quelli già visti.
- per gli editori - trovare plagio.
- Per scopi accademici - trovare plagio.
- Per raggruppare insieme file simile prima che vengano compressi.
- Confrontare revisioni di file su macchine diverse, ad esempio quelle di casa, del lavoro o portatili. Anche se quest'ultimo punto è stato superato dal controllo di versioni come Git.

2.3 Plague

Plague è l'implementazione dei suggerimenti trovati in [Whale, 1990]. Whale nel suo paper suggerisce le misure che possono essere usate per classificare i risultati e definisce quattro termini che possono essere usati nella sua valutazione:

- **Recall**: il numero di documenti rilevanti consultati come una proporzione di tutti i documenti rilevanti.
- **Precision(p)**: la proporzione dei documenti rilevanti nel gruppo consultato.

- **Sensitivity**: il limite affinché una data coppia di sottomissione venga nominata "simile" (abilità nel trovare somiglianze).
- **Selectivity**: l'abilità di mantenere un'alta precisione con l'aumentare della sensitivity (abilità nel rifiutare le differenze).

Un sistema di rilevamento dovrebbe permettere alla sensitivity di essere aggiustata, questo è simile a un sistema di recupero testi che permette all'utente di espandere o restringere una query. Questo permette uno scambio tra il numero di documenti e le somiglianze da aggiustare. Ci dovrebbe essere un limite dove la sensitivity selezionerà correttamente tutti i documenti plagiati e scarterà quelli dissimili.

Plague presenta alcuni problemi:

- È difficile da far adattare a nuovi linguaggi; richiede molto tempo.
- I risultati dati in output da Plague sono due liste ordinate dagli indici H e HT che hanno bisogno di essere interpretate. I risultati non sono immediatamente ovvi.
- Plague soffre di problemi di efficienza e fa affidamento su un numero addizionale di strumenti UNIX. Questo crea problemi di portabilità.

Complessivamente i risultati ottenuti sono buoni.

2.4 YAP

YAP ovvero "Yet Another Plague" è una serie di strumenti basata su Plague (2.3. Michael Wise creò la versione originale **YAP1** nel 1992 e seguì con una versione ottimizzata poco dopo chiamata **YAP2**. Nel 1996 Wise produsse la versione finale di YAP (YAP3) che lavorava in maniera molto efficiente con le sequenze trasposte. Lo scopo principale di YAP era di costruire sulle fondamenta di Plague, uno strumento che era ancora basato su tecniche di conteggio degli attributi e delle strutture, ma che superava alcuni dei problemi vissuti con queste tecniche.

Tutti I sistemi YAP lavorano essenzialmente nella stessa maniera e le operazioni si possono suddividere in due fasi:

1. La prima fase genera un token file per ciascuna sottomissione.
2. La seconda fase confronta coppie di token file.

I token rappresentano oggetti nel linguaggio scelto che possono essere sia strutture linguistiche oppure funzioni incorporate. Tutti gli identificatori sono costanti e vanno ignorati. Un piccolo assegnamento di solito ha 100-150 token mentre uno grande ne ha 400-700. La fase di tokenizzazione per ciascun linguaggio è eseguito separatamente, ma tutte le versioni hanno la stessa struttura:

1. Preprocessare i report sottomessi:
   (a) Rimuovere i commenti e le print-string.
   (b) Rimuovere le lettere non trovate negli identificatori legali.
   (c) Tradurre le lettere maiuscole in minuscole.
   (d) Formare una lista di token primitivi.
2. Cambiare i sinonimi alla forma comune. Questo è molto simile ad associare le parole ai loro iperonimi[1].
3. Identificare blocchi di funzioni/procedure.
4. Stampare blocchi di funzioni nell'ordine di chiamata. Blocchi di funzioni ripetuti vengono estesi una sola volta, le chiamate successive vengono rimpiazzate da un token numerato che rappresenta quella funzione. Questo previene un'esplosione nei token.
5. Riconoscere e stampare token che rappresentano parti della lingua di destinazione da un dato vocabolario. Le chiamate di funzioni vengono mappate a FUN e altri identificatori vengono ignorati.

YAP è capace di gestire:

- Cambiamenti di commenti o nella formattazione.
- Cambiamenti di identificatori.
- Cambiamenti nell'ordine degli operandi.
- Cambiamenti nei tipi di dato.
- Rimpiazzare espressioni con altre equivalenti. YAP anche su questo, come per i punti precenti, è immune oppure a volte registra una piccola differenza (1 token).
- Cambiamento nell'ordine delle statement indipendenti.
- Cambiamento della struttura di statement di iterazione.
- Cambiamento della struttura di statement di selezione.
- Rimpiazzare chiamate di procedura dal corpo della procedura.
- introduzione di statement non strutturati - questo crea 1 token di differenza.
- Combinare segmenti di programma originale con altri copiati.

YAP è accurato come Plague nel trovare alte somiglianze e nell'evitare di trovare risultati dove non esistono.

[1] iperonimia, ovvero quando un termine è incluso nel significato dell'altro (es: cinquecento - utilitaria -automobile)

2.5 JPlag

Il sistema JPlag [JPlag] trova somiglianze tra più set di file di codice sorgente. JPlag non si limita a confrontare byte di testo, ma è consapevole della sintassi del linguaggio di programmazione e della struttura del programma. Questa consapevolezza lo rende robusto contro i tentativi di mascherare le somiglianze tra i file plagiati. JPlag supporta *Java*, *C#*, *C*, *C++*, *Scheme* e il linguaggio naturale.

La forza di JPlag risiede anche nella potente interfaccia grafica per presentare i suoi risultati(Figure 2.2), che risulta essere simile a quella utilizzata da MOSS (3.2).

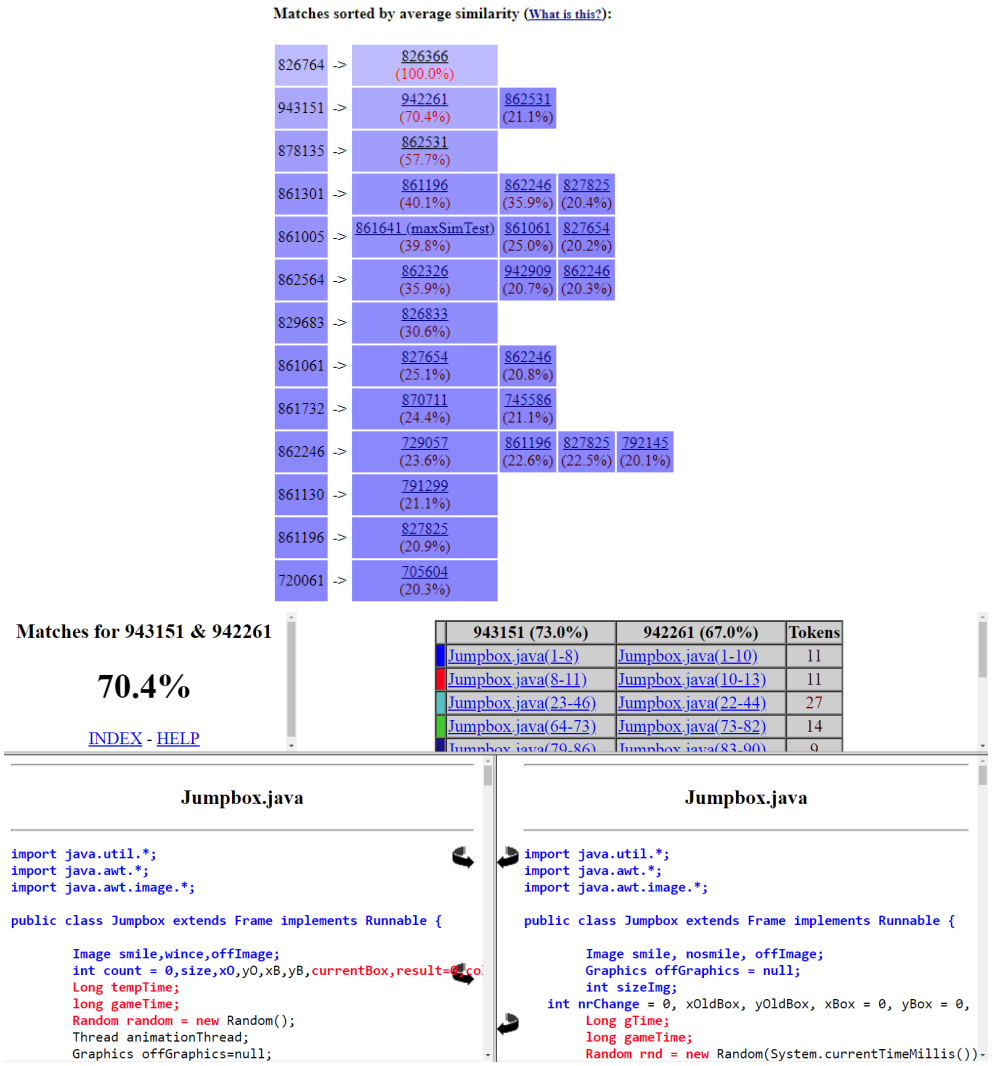

Figura 2.2: Esempio della visualizzazione dei risultati di JPlag e della pagina di analisi degli ultimi

3 | Studio ed implementazione

L'applicazione è stata sviluppata (3.1) utilizzando **Java**. Questa consente di accedere al proprio account GitHub, selezionare l'organizzazione utilizzata per Classroom e infine di scaricare le repositories degli studenti, le quali sono organizzate a loro volta in progetti corrispondenti agli assignment di GitHub Classroom.

Una volta selezionato il progetto da voler prendere in analisi è possibile selezionare le varie repositories contenenti il lavoro degli studenti per poi essere scaricate nella cartella che si desidera. Le repositories possono essere scaricate in due modalità *Salta* o *Aggiorna*. La prima consente di saltare le repository già scaricate in precedenza, mentre la seconda modalità consente di aggiornare all'ultimo commit effettuato nella repository.

Una volta scaricati i lavori che si vogliono analizzare si può eseguire il controllo del plagio. Esso è possibile tramite il tasto nella barra superiore dell'applicazione. Una volta cliccato il pulsante verrà richiesto di selezionare prima la cartella contenente le repositories da analizzare e poi la cartella contenente i files base, cioè il modello, dell'assignment. I significati di queste directory verranno analizzati in dettaglio nella sezione dedicata a MOSS (3.2).

Abbiamo scelto di utilizzare il servizio MOSS oltre per i suoi risultati soddisfacenti anche perché è un progetto molto attivo e in continuo aggiornamento, il che lo rende sicuramente la migliore scelta gratuita.

3.1 Implementazione

Come detto in precedenza il progetto è stato realizzato con l'ausilio di **Java**. Abbiamo scelto di utilizzare questo particolare linguaggio di programmazione poiché per prima cosa è cross-platform, cioè basta compilare una volta il programma per poi poter essere eseguito su qualsiasi sistema operativo avente una JVM[1]. Inoltre l'eccellente documentazione a corredo di questo linguaggio e delle sue librerie ha avuto un notevole rilievo, credo che il miglior strumento al mondo può essere inutilizzabile se non si hanno le istruzioni adeguate. Ovviamente anche la presenza di MOJI, la libreria Java di MOSS,

[1] JVM: Java Virtual Machine

ha influito nella scelta. Infine l'esperienza maturata negli anni nell'uso di questo linguaggio è stata decisiva nella scelta, consentendo di lavorare allo sviluppo in maniera agevole.

Fin da subito abbiamo optato per il *pattern architetturale* **MVC - Model View Controller** (3.1.1) così da rendere indipendenti i tre moduli grazie anche all'aggiunta di interfacce.

I moduli comunicano tra di loro grazie alle interfacce `IModel`, `IView` e `IController`. Questo implica che i vari moduli non conoscono come sono realizzati gli altri, ma sanno solo come è formata l'interfaccia, e quindi quali sono i "servizi" messi a disposizione, cioè i metodi offerti.

Potrebbe sembrare irrilevante questa scissione netta, ma in realtà ha molti vantaggi, tra cui permette ad esempio ad un team di sviluppo di dividersi in gruppi separati e ogni gruppo può occuparsi solo della sua sezione (Model, View, Controller), dando per assodata l'interfaccia fornita dagli altri gruppi di lavoro. Credo sia buona prassi strutturare ogni progetto in modo professionale e rendendolo fin dalla prima bozza il più scalabile possibile, così da rendere minime le modifiche e gli adattamenti se si avesse la necessità di espanderlo.

Inoltre un'altra accortezza è stata quella di utilizzare il *pattern creazionale* **Singleton**, il quale garantisce la creazione di un unico oggetto di una classe. In particolare è stato utilizzato per garantire che gli oggetti Model, View e Controller siano unici e quindi non ci sia ambiguità nel chiamarli. Ad esempio se non si fosse utilizzato questo pattern si sarebbero potuti creare due oggetti della stessa classe con il risultato che l'applicazione non funzionerebbe correttamente.

Di seguito in Figura 3.1 viene mostrato come interagiscono le interfacce e viene evidenziato il metodo `getInstance()` delle varie classi, il quale implementa il singleton e torna come oggetto l'istanza univoca della classe stessa. Farei anche notare che il costruttore è reso privato così da impedire la creazione di altri oggetti al di fuori da quello creato grazie al metodo appena introdotto.

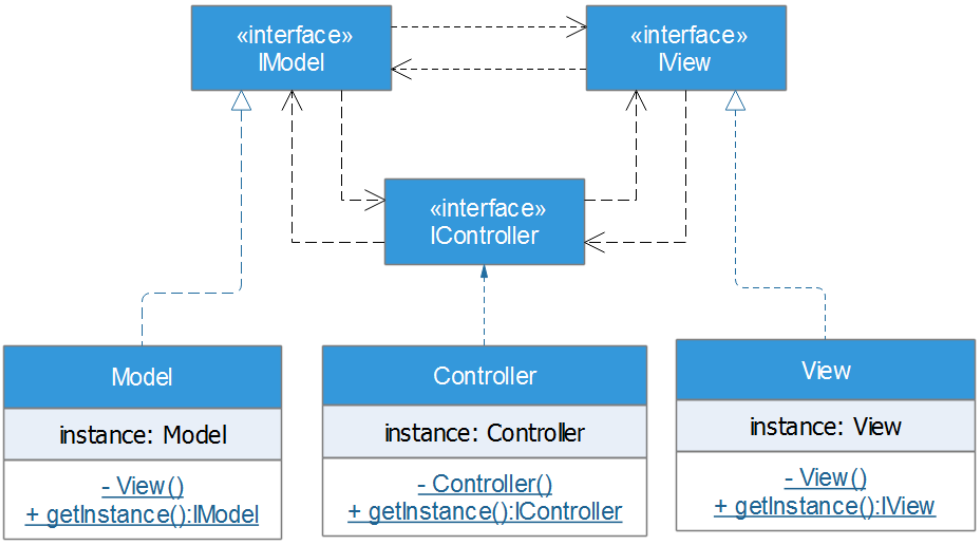

Figura 3.1: UML: Model View Controller

Ovviamente le tre classi cardine di tutta l'applicazione sono proprio il `Model`, il `View` ed il `Controller` che implementano tutti i metodi delle interfacce che gestiscono tutte le interazioni con le stesse.

Un'altra parte fondamentale del progetto è la gestione logica delle repositories degli studenti. Per implementare questa, si è optato per un'organizzazione gerarchica delle repository. Si può immaginare come una piramide con alla base le repository rappresentate dagli oggetti della classe `GitHubRepository`, il livello successivo è quello dei progetti rappresentati dalla gli oggetti della classe `GitHubProjecte` al vertice c'è l'organizzazione che racchiude tutto con la calasse `GitHubOrganization`. Di seguito è riportato uno schema delle Classi riguardanti GitHub con i principali metodi (Figura 3.2).

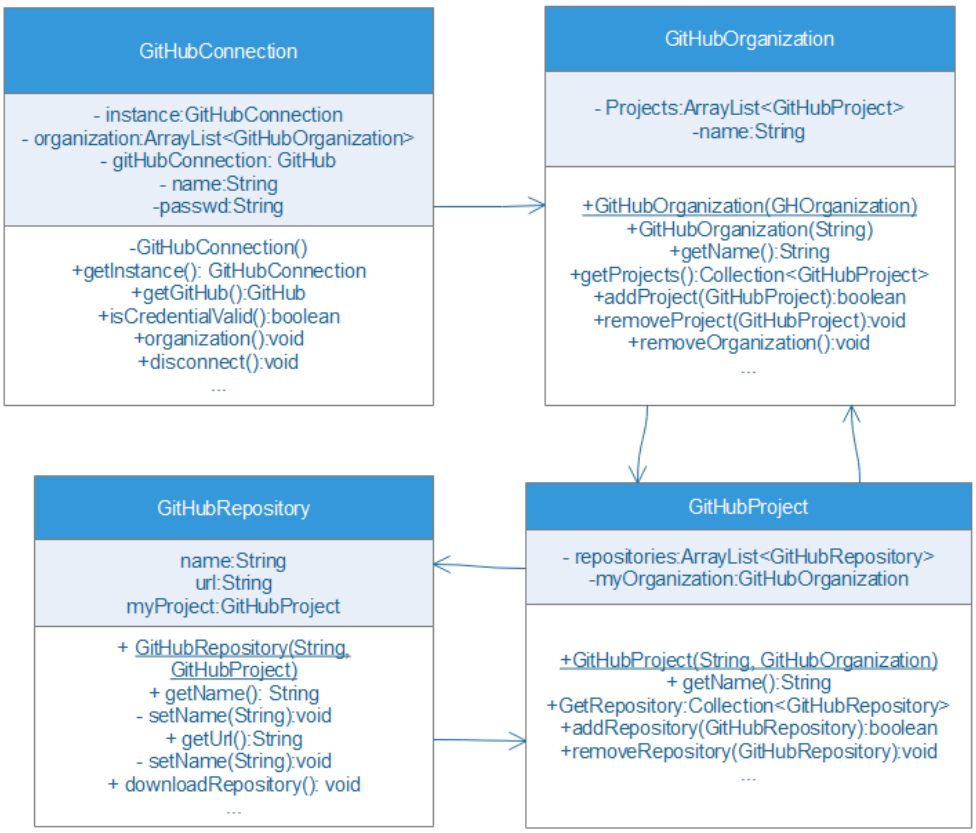

Figura 3.2: UML: GitHub Classes

Tutte le organizzazioni sono memorizzate nella classe `GitHubConnection`. Questa classe svolge un ruolo fondamentale nella connessione a GitHub. Grazie alla libreria GitHub API di [Kawaguchi] abbiamo potuto utilizzare le API di GitHub. Purtroppo ancora non esistono API specifiche per GitHub Classroom. La mancanza di queste API dedicate si è sentita sopratutto per trovare gli assignment, cioè i progetti. Per fare questa selezione si è dovuto applicare un filtro sui nomi, la parte comune di quest'ultimi è proprio il nome del progetto.

Le repository hanno il metodo `downloadRepository()` che permette la connessione alla repository di GitHub e di scaricare in locale tutto il materiale. Questo è stato possibile grazie alla libreria JGit che implementa tutte le funzionalità di Git.

L'altra parte indispensabile è quella della View. È stato scelto di renderla essenziale e senza fronzoli così da ottimizzare e facilitare l'utilizzo da parte dell'utente. Questa è stata strutturata in modo tale da dividere in due parti il JFrame, riservando la parte superiore alla connessione a GitHub, quindi l'inserimento delle credenziali. L'altra parte è dedicata alla selezione delle repository da scaricare.

Nella parte dedicata alle credenziali è presente una checkbox che permette di salvare il nome utente di GitHub così da agevolare l'accesso. Si è sceloto di non effettuare il salvataggio della password per motivi di sicurezza.

La parte della selezione delle repositories è abilitata esclusivamete dopo aver effettuato il login su GitHub. I due menù a tendina consentono di selezionare per prima cosa l'organizzazione e poi il progetto desiderato. I progetti e le repository sono caricate dinamicamente all'interno dell'applicazione, poichè come detto in preccedenza si devono analizzare tutte le repositories per determinare a quale progetto appartengono. Una volata selezionato il progetto, ad esempio l'appello dell'esame, comparirà la lista delle repositories che fanno parte di quello specifico progetto. Queste si potranno selezionare per poi scaricarle tramite l'apposito pulsante. Prima di scaricare si può selezionare la modalità di download:

- **Salta**: se trova una repository omonima nella cartella di download verrà consierata già scaricata.

- **Aggiorna**: il software si connette tramite Git e scarica la versione più recente della repository.

Una volta cliccato il pulsante di download l'applicazione chiederà all'utente di selezionare la directory dove verranno salvate le repositories. In linea con l'organizzazione data nel model, si è scelto di posizionare le repositories all'interno della cartella chiamata con il nome del progetto e a sua volta situata all'interno di un'altra cartella con il nome dell'organizzazione.

Nella barra superiore c'è la scelta della lingua (italiano e inglese) e la sezione dedicata a MOSS, la quale è attivabile cliccando nel tasto Start. Una volta cliccato il pulsante verrà chiesto prima di selezionare la directory contenente i progetti da analizzare e successivamente di selezionare la directory del modello. Dopo qualche secondo, al massimo pochi minuti apparirà un pop-up contenente il link del risultato dell'analisi effettuata da MOSS.

Di seguito è riportata una preview dell'interfaccia grafica (Figura 3.3).

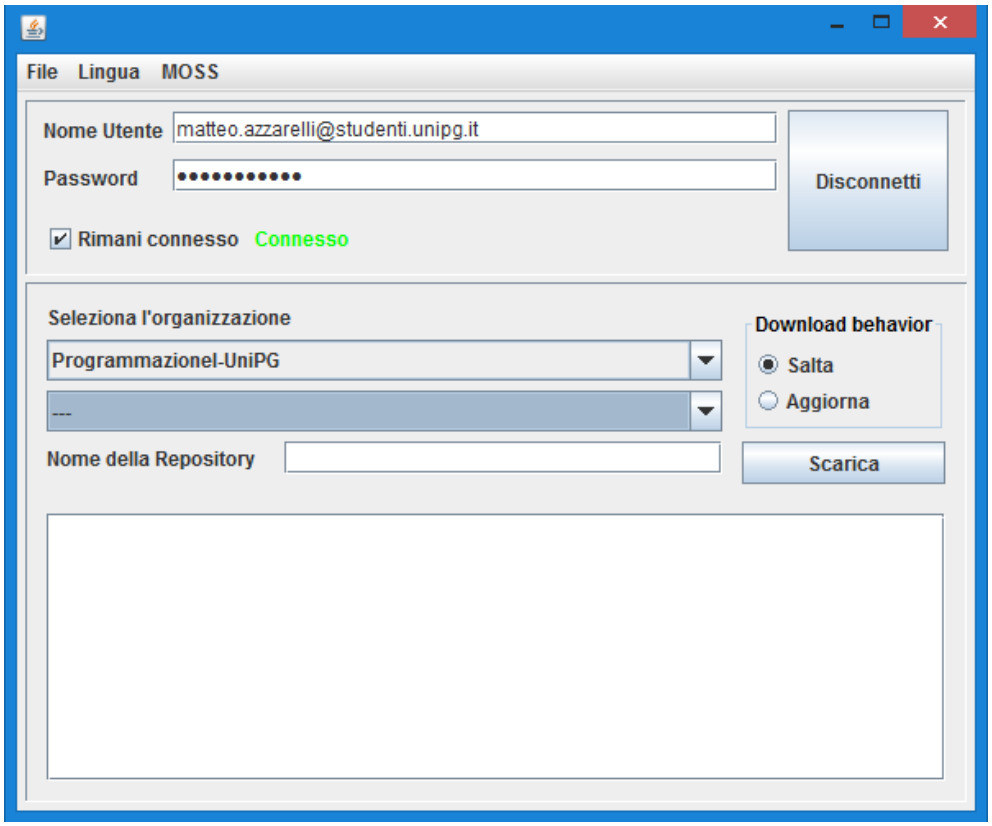

Figura 3.3: Interfaccia grafica dell'applicazione

Come già detto in precedenza l'applicazione segue il pattern Model View Controller seguendo le linee guida come spiegato nella sezione successiva 3.1.1.

3.1.1 MVC - Model View Controller

I *pattern architetturali* definiscono il più alto livello di astrazione software. Esso infatti descrive la struttura di un sistema software in termini di relazioni e di interazioni che possono essere coinvolte tra i sottoinsiemi della struttura.

È bene scegliere attentamente quale pattern utilizzare poiché esso influenzerà notevolmente le fasi di progettazione e soprattutto di realizzazione.

Il **MVC - Model View Controller** [Russo and Mecella] è un pattern architetturale specializzato nella progettazione e strutturazione modulare di applicazioni software interattive. È stato ideato da *Trygve Reenskaug* (sviluppatore Smalltalk presso lo Xerox Palo Alto Reserch Center) nel 1979.

Ecco le parti di cui è composto:

- **Model**: modello dei dati dell'applicazione. Esso incapsula lo stato dell'applicazione e gestisce l'accesso ai dati. Gli stati poi vengono aggior-

nati e una volta avvenuto il cambiamento, viene mandata una notifica al view.

- **Controller**: logica di controllo che gestisce il modello dei dati. L'interazione da parte dell'utente è permessa grazie agli eventi e ai comandi generati dal Controller, che andranno a modificare il model.
- **View**: rappresentazione grafica e interattiva del model. Esso definisce le modalità di presentazione dei dati dello stato dell'applicazione. È una rappresentazione anche interattiva dato che consente proprio l'interazione con l'utente. Riceve delle notifiche dal Model se avvengono dei cambiamenti e poi successivamente aggiorna la visualizzazione.

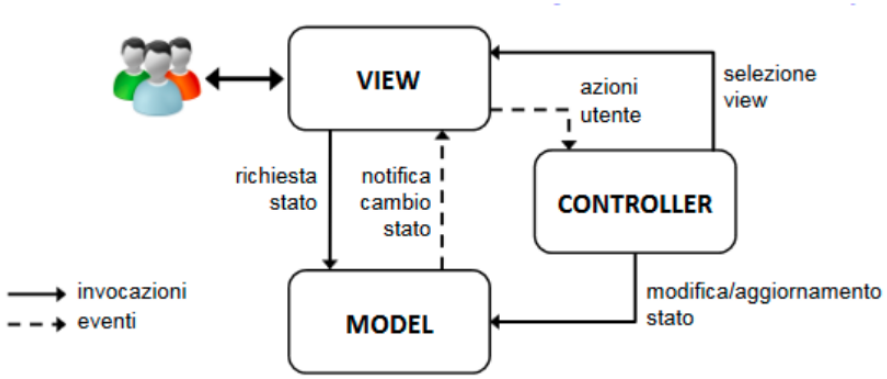

Figura 3.4: UML: Schema generico del pattern MVC

Inizializzazione dell'applicazione:

1. Viene creato il model.
2. Viene creato il view il quale richiama il metodo getIstance() del model.
3. Viene creato il controller il quale chiama il metodo getIstance() del Model e della View.
4. Il view si registra come listener (o observer) del model per ricevere notifiche di aggiornamento dal model (observable).
5. Il controller si registra come listener (o observer) del view per ricevere dal view (observable) gli eventi generati dall'utente.

Interazione da parte dell'utente con l'applicazione:

- Il view riconosce l'azione dell'utente (ad esempio, premere un bottone) e notifica il controller registrato come listener.
- Il controller interagisce con il model per realizzare l'azione richiesta e aggiornare/modificare lo stato o i dati.

- Il model notifica al view registrato come listener le modifiche e gli aggiornamenti.

- Il view in base al nuovo stato, aggiorna la visualizzazione. Il nuovo stato può essere ottenuto tramite:

  - Approccio **push**: il model notifica al view sia il cambiamento di stato che le informazioni generate. Approccio utilizzato nella nostra applicazione.

  - Approccio **pull**: il view riceve dal model la notifica del cambiamento di stato e poi accede al model per ottenere le informazioni aggiornate.

Nel paradigma MVC l'interazione tra componenti è basata su meccanismi di propagazione e gestione di eventi

L'interazione tra il view e il controller in Java avviene in base al meccanismo di propagazione e gestione eventi Swing/AWT, i componenti controller sono EventListener (es MouseListener) associati ai componenti grafici view (es JButton). Diversamente avviene invece l'interazione tra model e view: il tutto viene gestito tramite la filosofia Observer-Observable, implementato tramite la gestione di variabili con metodi `get` e `set`, i quali effettuano dei controlli e attivano dei meccanismi di feedback, comunicazione tra le due classi attraverso la chiamata di metodi, si gestiscono gli eventi.

Esempio del funzionamento del bottone *Connect*. Quando l'utente clicca il pulsante connect si attiva l'evento associato. Questo viene raccolto dalla classe `CredentialJPanelController` che a sua volta procede a verificare le credenziali chiamando il metodo *isCredentialValidGitHub()* messo a disposizione dell'interfaccia `IModel`. Una volta sincerate le credenziali si passa alla connessione vera e propria, sempre messa a disposizione dal'interfaccia `IModel` tramite il metodo *autenticateToGitHub()*. Se le credenziali non dovessero andare bene verrebbe chiamato il metodo *disconnectGitHub()* che si occuperebbe di notificare l'impossibilità di connettersi.

3.2 MOSS

MOSS è un sistema automatico per determinare la similarità dei sorgenti di programmi [Aiken, 1994-1997].

MOSS sta per Measure Of Software Similarity, accetta gruppi di documenti e ritorna un insieme di pagine HTML che mostrano, dove significanti, sezioni di coppie di documenti molto simili. MOSS è principalmente usato per *scovare plagi* in progetti di programmazione in informatica e altri corsi di ingegneria, sebbene comunque supporti molti altri formati di testo.

3.2.1 Funzionamento

Come descritto nel paper [Clough, 2000], il servizio usa una robusta selezione, cioè sceglie poche fingerprint[2] 3.2.1 per la stessa qualità di risultati, rendendolo più efficiente e scalabile rispetto ad altri algoritmi.

In MOSS, l'informazione posizionale, cioè il documento e il numero di linea, è memorizzato per ciascuna fingerprint selezionata.

Il *primo passo* dell'algoritmo costruisce un indice che mappa le fingerprint alle locazioni per tutti i documenti, come l'indice invertito costruito dai motori di ricerca per mappare parole a posizioni nei documenti. Nel *secondo passo*, per ciascun documento viene effettuata la fingerprint una seconda volta e le fingerprint selezionate vengono cercate nell'indice; questo ritorna la lista di tutte le fingerprint risultanti per ciascun documento.

Ora la lista delle fingerprint risultanti per un documento d possono contenere le fingerprint di molti documenti differenti *d1,d2,...* . Al prossimo passo, la lista delle fingerprint risultanti per ciascun documento d viene ordinata per documento e i risultati vengono raggruppati per ciascun paio di documenti (d,d1),(d,d2).

Le coppie di documenti vengono ordinati per rank dato dalla dimensione (numero di fingerprint) e i più grandi vengono riportati all'utente, cioò i documenti con maggior numero di plagi saranno visualizzati per primi.

Notate che fino a quest'ultimo passo, non è stata richiesta nessuna coppia. Questo è molto importante, poiché non potremmo sperare di effettuare il rilevamento di copie confrontando ciascuna coppia di documenti in un grande corpus, rendendolo per questo molto efficiente.

MOSS ha varie centinaia di utenti che sperano di trovare plagi in una moltitudine di dati di vari tipi. Esso richiede di specificare il tipo di documenti che andrà ad analizzare. Questo perché per ciascun formato di documenti, provvede ad eliminare le caratteristiche che non dovrebbero distinguere i vari documenti (ad esempio, si elimina gli spazi bianchi dal testo). Questo documento generato da questa scrematura è l'input da dare al motore fingerprint, il quale non sa niente riguardo i differenti tipi di documento, caratteristica che lo rende fortemente scalabile.

Fingerprints

Individuare copie parziali di file è molto più complesso rispetto a trovare delle copie complete.

In generale molte delle tecniche che trovano le copie parziali seguono la seguente idea.

Si divida un documento in **k-grams**[3], dove k è un parametro scelto

[2] In informatica un *algoritmo di fingerprinting* è una procedura che mappa un elemento di dati arbitrariamente grande in una stringa di bit molto più breve che identifica in modo univoco i dati di origine.

[3] Un k-gram è una sottostringa contigua di lunghezza k.

A do run run run, a do run run

(a) un po di testo [The Crystals. Da do run run, 1963]

adorunrunrunadorunrun

(b) il testo senza alcune caratteristiche irrilevanti

adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun

(c) la sequenza di 5-grams derivata dal testo

Figura 3.6: Alcuni esempi di fingerprinting

dall'utente.

Per esempio, la Figura 3.6.c contiene tutti i 5-gram della stringa di caratteri della Figura 3.6.b derivata dalla stringa del testo di una nota canzone (Figura 3.6.a).

Notare che ci sono almeno tanti k-gram quanti sono i caratteri nel documento, cioè ogni posizione nel documento (eccetto per le ultime k-1 posizioni) segnano l'inizio di un k-gram.

A questo punto utilizza la funzione hash per ciascun k-gram e selezioniamo tra essi qualche sottoinsieme: questi andranno a formare i fingerprint.

L'insieme di **fingerprint** è un piccolo sottoinsieme dell'insieme di tutti i "k-grams hashes" .

Un fingerprint inoltre contiene informazioni posizionali, descrivendo il documento e la locazione all'interno del documento dal quale il fingerprint è stato ricavato.

Se la funzione hash è stata scelta in modo tale che la probabilità di collisioni è molto bassa, allora ogni volta che due documenti condividono uno o più fingerprint, è molto probabile che loro condividono un k-gram.

3.2.2 Il servizio

MOSS mette a disposizione un web services accessibile tramite vari script e librerie implementate negli anni dagli sviluppatori di tutto il mondo. In particolare nel progetto realizzato è stata usata la libreria java **MOJI** [Zielke], la quale permette di implementare agevolmente un client per interfacciarsi con il servizio offerto da MOSS.

Per prima cosa bisogna registrarsi al servizio inviando una e-mail ll'indirizzo `moss@moss.stanford.edu` con oggetto vuoto e con corpo dell'email come segue:

```
registeruser mail username@domain
```

dove l'ultima parte in italics è il tuo indirizzo email. Dopo pochi minuti viene ricevuta nella mail indicata le istruzioni sul funzionamento di MOSS e cercando *$userid* si troverà il token di autenticazione per poter utilizzare il servizio.

Questa libreria permette di inviare le repositories da controllare in blocco, cioè passando come parametro una cartella strutturata come è mostrato successivamente.

```
solution_directory
|- student1
    |- main.c
    |- ...
|- student2
    |- ...
|_ student3
    |- ...
```

Oltre alle repositories da analizzare è anche possibile inviare una cartella contenente la traccia base. Quest'ultima è l'insieme dei file dati dal professore agli studenti ed è utilizzata per ridurre ulteriormente la possibilità di errori nella valutazione dei plagi e quindi aumentare l'affidabilità dei risultati.

MOSS da là possibilità di settare altri parametri come:

- **optC**: è un commento che apparirà nella pagina HTML di ritorno di MOSS.
- **optD**: indica che l'invio è strutturato a cartelle e non a file. In MOJI c'è solo la versione a Directory.
- **optM**: una porzione di codi quanto specificato. Fornisce un metodo semplice per rilevare il codice base/modello.
- **optN**: definisce quante coppie si devono mostrare nella pagina dei risultai.
- **optX**: supporto alle fatures sperimentali. Settare a 1 oer usare il server sperimentale di MOSS.

Dopo pochi secondi che si sono inviati i documenti da analizzare verrà ritornato un URL che porta alla pagina HTML con i risultai (Figura 3.7). I risultati rimarranno nei server di MOSS per circa 14 giorni dopo l'invio della richiesta. All'interno della pagina dei risultati si troveranno le coppie dei progetti incriminati con la rispettiva percentuale di codice simile e il numero righe di codice sospette. La percentuale dei risultati indica il grado di sovrapposizione tra il progetto stesso e quello con cui si è comparato.


Moss Results

Wed Jul 4 13:31:37 PDT 2018

Options -l c -d -m 10

ESAME PROGRAMMAZIONE 1

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

File 1	File 2	Lines Matched
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]c98/ (96%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]82/ (97%)	1035
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]ni4/ (74%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]elli/ (78%)	776
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]ce/ (96%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]rdo/ (97%)	669
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]a/ (85%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]p0/ (92%)	831
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]etta/ (84%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]etti/ (86%)	759
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]etto/ (76%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]sae/ (80%)	722
C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]c0/ (80%)	C:/ProgrammazioneI-UniPG/prova-finale-di-programmazione-secondo-appello/[redacted]sae/ (78%)	729

Figura 3.7: Esempio di risultati di MOSS

Aprendo uno dei risultati verranno mostrati i sorgenti a confronto (Figura 3.8) e nella parte alta della pagina HTML verranno visualizzati in vari colori le parti di codice simili.

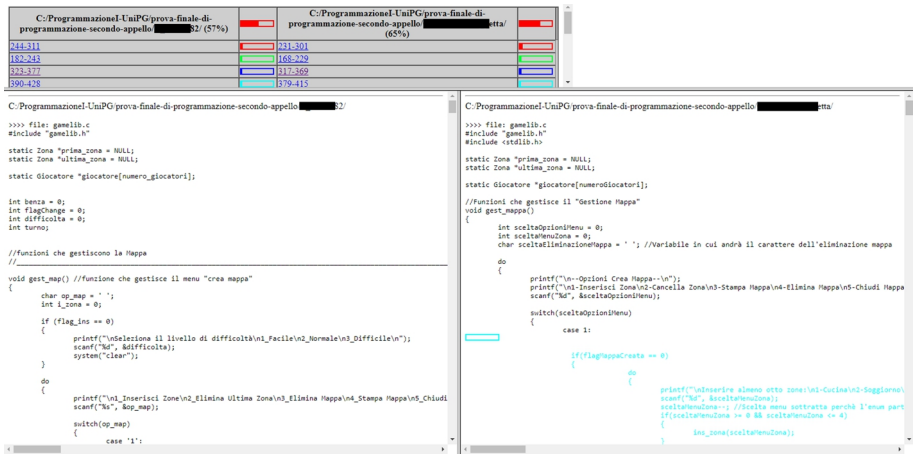

Figura 3.8: Esempio pagina analisi sospetto plagio di MOSS

Un'altro aspetto da tenere in considerazione è la privacy e quindi sono state prese delle precauzioni per mantenere private le pagine dei risultati. Per prima cosa i risultati non possono essere scansionati da robots o sfogliati da persone che navigano nel Web. Inoltre il numero casuale nell'URL viene reso noto solo all'account che ha inviato la query e non vi è alcun modo per accedere ai risultati tranne che attraverso quell'URL. Per concludere, le pagine dei risultati scadono automaticamente dopo 14 giorni e il codice non viene mantenuto nel server. L'unione di queste accortezze rende molto basso il potenziale di abuso.

Conclusioni

Dopo aver visto le potenzialità del software Git e della piattaforma che offre un eccellente servizio gratuito come GitHub, siamo passati all'analisi dei vari software ideati per scovare similitudini all'interno di progetti software e scoraggiare i tentativi di copiatura. Dopo questo importante studio abbiamo descritto l'implementazione del programma da noi sviluppato riportando i passaggi ritenuti più interessanti. Da questo siamo passati alla descrizione del servizio indispensabile di MOSS, così da poter prendere coscienza su cosa stiamo utilizzando e fornendo un supporto alla comprensione dei risultati.

In una fase di test dell'applicazione abbiamo potuto costatare l'effettivo funzionamento ed un'efficacia strabiliante nel trovare i plagi. Quindi ci riteniamo soddisfatti del lavoro fatto, ma sempre pensando a dei miglioramenti futuri.

Questa è un'applicazione che può essere ampliata a piacimento e in particolare potrebbe essere utile aggiungere le seguenti features:

- Download dei risultati di MOSS (dopo 14 giorni vengono eliminati dal server di MOSS).
- Aggiungere una parte sull'analisi statica dei progetti (es. Valgrind per progetti in `C`), per fornire aiuto al docente con una pre-valutazione del progetto.
- Ottenere il nome e il cognome degli studenti dal file ReadMe delle repository e costruire dinamicamente un gestionale degli studenti.
- ...

Bibliografia

Alex Aiken. Moss: Asystem for detectiong software similarity, 1994-1997. URL https://theory.stanford.edu/~aiken/moss/.

S. Chacon and B. Straub. *Pro Git*. The expert's voice. Apress, 2018. URL http://git-scm.com/book/en/v2.

Paul Clough. Plagiarism in natural and programming languages: an overview of current tools and technologies, 2000. URL https://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf.

J. A. W. Faidhi and S. K. Robinson. An empirical approach for detecting program similarity and plagiarism within a university programming environment. *Comput. Educ.*, 11(1):11–19, January 1987. ISSN 0360-1315.

Dick Grune. The software and text similarity tester sim. URL https://dickgrune.com/Programs/similarity_tester/.

JPlag. Jplag. URL https://jplag.ipd.kit.edu/.

Kohsuke Kawaguchi. Github api. URL http://github-api.kohsuke.org/.

Alan Parker and James O. Hamblen. Computer algorithms for plagiarism detection. *Education, IEEE Transactions on*, 32:94–99, June 1989.

Alessandro Russo and Massimo Mecella. Esercitazioni di progettazione del software. URL http://www.dis.uniroma1.it/~mecella/didattica/2013/EsercitazioniProgettazioneSoftware/E08/Slide_MVC_StrutturaApplicazioni.pdf.

G. Whale. Identification of program similarity in large populations. *The Computer Journal*, 33(2):140–146, 1990. URL http://dx.doi.org/10.1093/comjnl/33.2.140.

Björn Zielke. Moji client for the moss plagiarism detection service. URL https://zielke.it/moji/.