
Progetto di Ingegneria del Software

Relazione di progetto

Giovanni Dini - gioggi2002@gmail.com - g.dini3@campus.uniurb.it

*Matricola 232274 • Università degli Studi di Urbino "Carlo Bo" - Facoltà di Informatica Applicata • Sessione
Autunnale 2012*

Specifica del problema

Il progetto d'esame che si è deciso di realizzare consiste in un sistema di crawlers che raccolgono dati da pagine web.

In particolare, l'attuale implementazione può essere considerata quasi uno "spambot", dato che il suo principale scopo è quello di raccogliere indirizzi email, salvandoli in un file con formato csv, pronto per l'esportazione ad altri servizi (es.: Gmail).

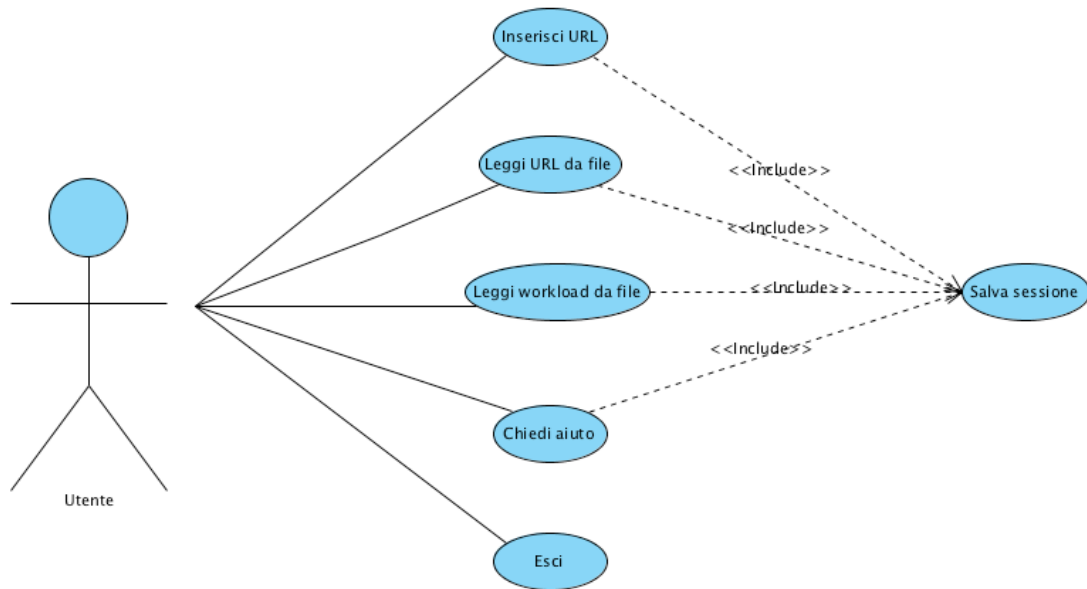
Il software offre comode funzioni di input/output su files, al pari di funzioni di ripresa da sessioni precedenti e salvataggio per utilizzo in sessioni successive.

Il comportamento del programma è personalizzabile tramite un semplice file di configurazione (vedasi sezione "scelte di progetto").

L'intero progetto con i passi della sua realizzazione è disponibile online all'indirizzo <https://github.com/gioggi2002/ProgettoIngSoft>. La licenza di distribuzione di questo software è la Creative Commons 3.0. Il software può essere modificato e ridistribuito solo con attribuzione del lavoro originale al suo autore e non per scopi commerciali.

Specifica dei requisiti

L'attore presente nel diagramma UML degli Use Cases è uno solo, ovvero l'utente che avvia il programma.



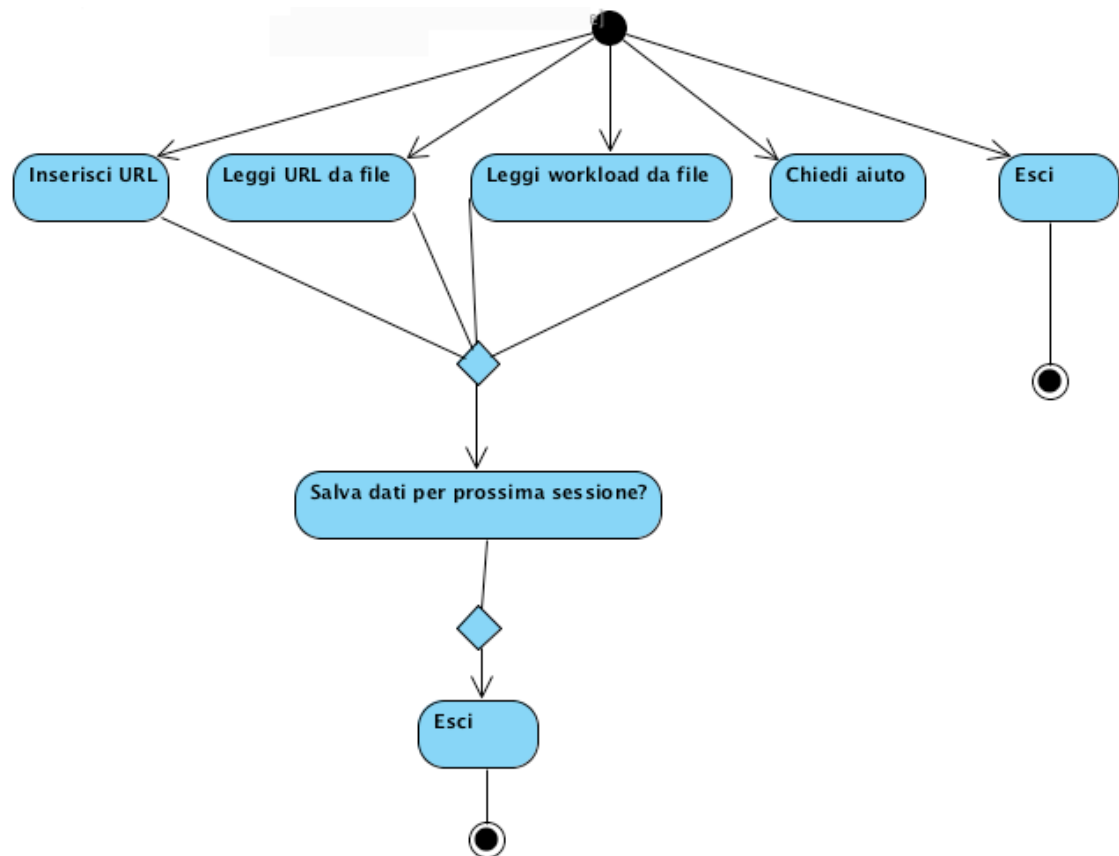
Use case	Inserisci URL
Actor	User
Preconditions	
Basic course of events	Lo User inserisce l'URL da analizzare
Postconditions	Il programma analizza l'URL
Alternative paths	L'URL viene letto da file/La workload viene letta da file/Lo User chiede aiuto/Lo User esce

Use case	Leggi URL(s) da file
Actor	User
Preconditions	
Basic course of events	Il programma legge l'URL(s) dal file workload.csv
Postconditions	Il programma analizza l'URL
Alternative paths	Lo User inserisce l'URL/La workload viene letta da file/Lo User chiede aiuto/Lo User esce

Use case	La workload viene letta da file
Actor	User
Preconditions	
Basic course of events	Il programma legge la workload dal file nextWorkload.csv
Postconditions	Il programma analizza l'URL
Alternative paths	Lo User inserisce l'URL/L'URL viene letto da file/Lo User chiede aiuto/Lo User esce

Use case	Chiedi aiuto
Actor	User
Preconditions	
Basic course of events	Vengono stampati i messaggi di aiuto
Postconditions	Il programma stampa i messaggi di aiuto, dopodiché ripropone allo User le scelte precedenti
Alternative paths	Lo User inserisce l'URL/L'URL viene letto da file/La workload viene letta da file/Lo User esce

DIAGRAMMA DELLE ATTIVITÀ:

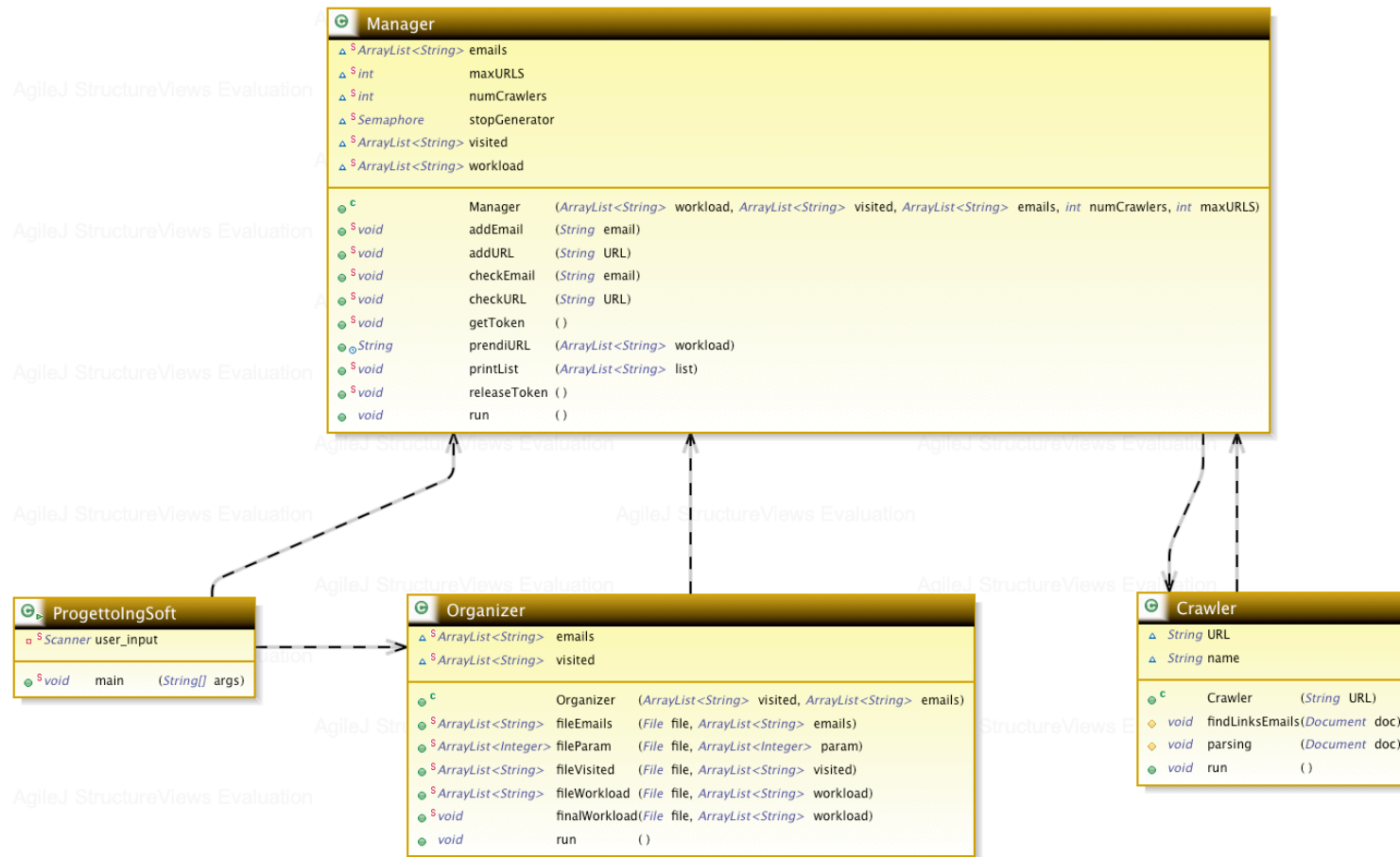


Analisi e progettazione

SCELTE DI PROGETTO:

- Il formato dei files di input/output è csv (Comma Separated Values). Si è scelto di utilizzare questo formato perché facilmente integrabile con altri servizi (es.: è sufficiente copiare il contenuto del file email.csv su Gmail per inviare rapidamente a tutti gli indirizzi contenuti nel file csv) e facilmente modificabile.
- Anche le funzioni per salvare/riprendere una sessione sono costruite sull'utilizzo files csv.
- Il file di configurazione (preferences.cfg) è costituito semplicemente da due parametri. Il primo indica il numero massimo di crawlers che possono operare in contemporanea, mentre il secondo indica la profondità dell'analisi, espressa con il numero massimo di URL che devono essere analizzati in una singola sessione.
- Le liste da utilizzare sono di tipo ArrayList. Si è scelto di utilizzare queste e non semplici Array perché, a differenza di questi ultimi, non hanno una dimensione fissa e questa può essere modificata dinamicamente durante l'esecuzione. Il problema che si poneva a questo punto, però, era la necessità di sincronia durante l'utilizzo, cosa che era fornita solo da Collections di tipo Vector. I Vector offrivano sincronia ma prestazioni sensibilmente peggiori, inoltre le loro dimensioni non erano dinamiche ma andavano fissate di volta in volta. Per ovviare a questo problema, si è deciso di non lasciare che ogni crawler andasse a modificare le varie liste, ma che passasse invece dal manager.

Implementazione delle classi UML



Scelte progettuali relative alle classi e alle variabili



PROGETTOINGSOFT.JAVA (MAIN):

Questa classe inizializza le liste che saranno passate ai vari oggetti durante l'esecuzione (workload, visited, emails, param).

Propone poi il menu all'utente consentendogli di inserire l'URL di partenza dell'analisi, riprendere una sessione precedente, leggere una lista di URL da analizzare sempre tramite file, cancellare i contenuti dei file che contengono i dati delle sessioni precedenti (reset) o uscire.

Se uno dei files necessari non esiste, questa classe provvederà a crearlo in autonomia.

Si occupa di leggere i dati dal file di configurazione e istanzia il manager con i giusti parametri.

Attesa la terminazione del manager, istanzia l'organizer e attende anche la sua terminazione.

A questo punto chiede all'utente se vuole memorizzare gli URLs contenuti nella lista workload per un utilizzo successivo nelle prossime sessioni, dopodiché termina la sua esecuzione.

Manager		
ArrayList<String>	emails	
int	maxURLS	
int	numCrawlers	
Semaphore	stopGenerator	
ArrayList<String>	visited	
ArrayList<String>	workload	
c	Manager	(ArrayList<String> workload, ArrayList<String> visited, ArrayList<String> emails, int numCrawlers, int maxURLS)
void	addEmail	(String email)
void	addURL	(String URL)
void	checkEmail	(String email)
void	checkURL	(String URL)
void	getToken	()
String	prendiURL	(ArrayList<String> workload)
void	printList	(ArrayList<String> list)
void	releaseToken	()
void	run	()

MANAGER.JAVA:

Questa classe istanzia i vari crawler passandogli il giusto URL da analizzare.

Per cominciare istanzia un singolo crawler (Crawler_start), per analizzare l'URL che è stato passato direttamente dalla main. Questo crawler troverà poi (presumibilmente) altri URLs che, inseriti nella lista workload, verranno poi processati dai successivi crawlers.

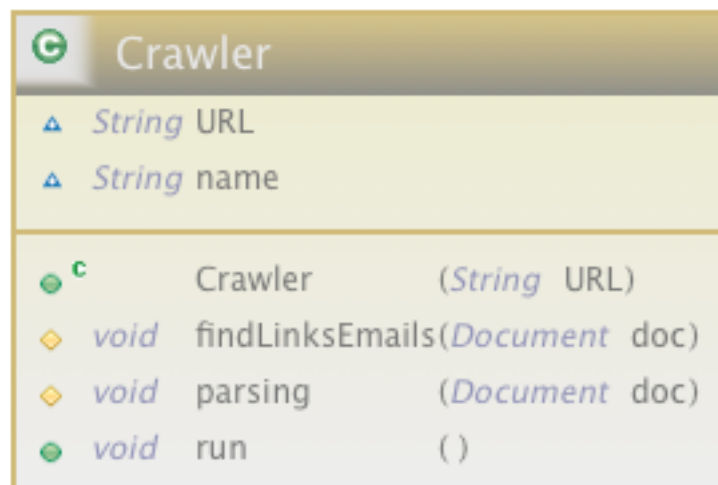
A questo punto istanzia nuovi crawlers fin quando non è stato raggiunto il numero massimo di crawlers contemporanei in esecuzione (o anche il numero massimo di crawlers necessari - es.: se sono contenuti solo 4 URLs nella lista workload e il numero massimo di crawlers contemporanei permessi è 8, saranno comunque generati solo 4 crawlers, ovviamente) o fin quando non è stato raggiunto il limite di profondità dell'analisi.

I metodi forniti da questa classe sono:

prendiURL: serve per prendere il primo URL nella workload. Una volta preso, questo viene poi aggiunto alla lista visited, per evitare che possa essere analizzato di nuovo.

- printList: è un metodo di debug utilizzato per stampare il contenuto della lista. Essendo, per l'appunto, di debug non è utilizzato, se non in stampe di controllo (al momento commentate).
- checkURL e checkEmail: sono metodi che effettuano il controllo degli URL e degli indirizzi email trovati dal crawler. Se sono già presenti nelle rispettive liste (visited e emails) non vengono aggiunti alle liste workload e emails.

- addURL e addEmail: questi metodi aggiungono l'URL o l'indirizzo email alle rispettive liste (visited e emails), dato che non risultano come già processati.
- getToken: all'avvio di un crawler, questo prende un token dal semaforo, per limitare il numero di crawlers contemporanei. Questo metodo serve, per l'appunto, a prendere un token dal semaforo.
- releaseToken: questo metodo effettua il procedimento inverso del precedente.



CRAWLER.JAVA:

Questa classe serve per la connessione alle pagine web, il loro parsing e la raccolta di indirizzi email e URLs.

Per prima cosa controlla se l'URL che gli è stato fornito è valido.

Poi effettua un tentativo di connessione, ricevendo lo status code della pagina e il suo tipo di contenuto. Se lo status code è diverso da 200 (connessione accettata), non vengono fatti ulteriori tentativi di connessione. Se lo status code è 200 viene effettuato il controllo al content-type della pagina, dopo aver filtrato le informazioni sulla codifica (tramite lo split con il carattere “;”). Se il tipo di pagina è diverso da text (es.: pdf), non viene effettuato il parsing del documento e la sua successiva analisi.

Durante il parsing viene fatta la ricerca degli indirizzi email e degli URLs. Entrambi vengono cercati allo stesso modo e selezionati grazie al tag href. I due vengono distinti per il fatto che gli indirizzi email cominciano con “mailto:”. Grazie a questo elemento è possibile distinguere gli URLs dagli indirizzi email. Durante l'analisi degli indirizzi email, però, è facile trovare indirizzi che includano opzioni per mailto (es.: <mailto:indirizzo@abcde.com?subject>). Per evitare questa eventualità, viene eliminato tutto ciò che si trova dopo il carattere “?”.

Organizer		
ArrayList<String>	emails	
ArrayList<String>	visited	
Organizer		(ArrayList<String> visited, ArrayList<String> emails)
ArrayList<String>	fileEmails	(File file, ArrayList<String> emails)
ArrayList<Integer>	fileParam	(File file, ArrayList<Integer> param)
ArrayList<String>	fileVisited	(File file, ArrayList<String> visited)
ArrayList<String>	fileWorkload	(File file, ArrayList<String> workload)
void	finalWorkload	(File file, ArrayList<String> workload)
void	run	()

ORGANIZER.JAVA:

Questa classe serve per le opzioni di input/output su files.

Quando viene invocata salva prima il contenuto della lista emails, poi della lista visited, nei rispettivi files.

I metodi forniti da questa classe sono:

- fileParam: legge e importa i parametri dal file di configurazione.
- fileWorkload: legge e importa gli URLs da visitare nella workload.
- fileEmails: legge e importa le emails nella lista emails.
- fileVisited: legge e importa gli URLs già visitati nella lista visited.
- finalWorkload: legge ed esporta gli URLs contenuti nella workload al momento della terminazione del programma in un file apposito.

LISTE:

- Lista workload: contiene tutti gli URL ancora in attesa di analisi.
- Lista visited: contiene tutti gli URL che sono già stati analizzati.
- Lista emails: contiene tutti gli indirizzi email trovati.
- Lista param: contiene i parametri di configurazione del programma.

Implementazione

PROGETTOINGSOFT.JAVA:

```
package progettoingsoft;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ProgettoIngSoft {

    // user_input è lo Scanner usato per ricevere i comandi dall'utente
    private static Scanner user_input;
```

```

/**
 * @param args
 */
public static void main(String[] args) {

    /**
     * Il thread main nasce.
     */

    // System.out.println("Thread main partito.");

    /**
     * Apro il menu iniziale, con le diverse scelte a disposizione
     * dell'utente.
     */

    System.out.println("Progetto di Ingegneria del Software\n");
    System.out.println("Autore: Giovanni Dini");
    System.out.println("Matricola: 232274");
    System.out.println("Email: gioggi2002@gmail.com\n");

```

```

        System.out.println("Operazioni disponibili:");
        System.out.print("- Inserire l'URL da cui cominciare l'analisi \n- Digitare \"file\" se ");
        System.out.print("si desidera leggere la workload dal file apposito (workload.csv)");
        System.out.println("- Digitare \"restart\" se si desidera riprendere una vecchia
sessione :");
        System.out.println("(digitare \"help\" per istruzioni o \"exit\" per uscire:");
        user_input = new Scanner(System.in);
        String first_url;
        first_url = user_input.next();

        /**
         * Sezione di help
         */

        while ("help".equals(first_url)) {
            System.out.print("\n\n*-----*\n\nCosa è e a cosa serve questo software?");
            System.out.print("\n\nQuesto software genera un sistema di crawler che girano per la
rete, ");
            System.out.print("raccogliendo indirizzi email e salvandoli per un successivo
utilizzo.");
            System.out.print("\n\n*-----*\n\nCome si utilizza?");
            System.out.print("\n\nIl funzionamento è molto semplice: è sufficiente inserire un URL
");

```

```

        System.out.print("(nel formato http://www.sito.com) e il programma lo analizzerà,
raccogliendo ");

        System.out.print("automaticamente gli indirizzi dei link che troverà e analizzando
successivamente ");

        System.out.print("anche quelli. Tutti gli indirizzi email trovati durante l'analisi sono
salvati per ");

        System.out.print("un successivo utilizzo. ");

        System.out.print("\n\n*-----*\n\nQuale è l'utilità pratica di questo software?");

        System.out.print("\n\nIn questa versione il software si limita a raccogliere indirizzi
email ");

        System.out.print("(si ricorda che lo spam non è una pratica legale), ma con poche e
semplici ");

        System.out.print("modifiche è possibile fargli raccogliere e analizzare praticamente
tutto ciò che ");

        System.out.print("può essere contenuto in una pagina web. Vista la natura di questo
software ");

        System.out.print("(realizzato per un progetto d'esame), è da considerarsi un puro
esercizio, ma che ");

        System.out.print("con poche modifiche può avere scopi decisamente più utili.");

        System.out.print("\n\n*-----*\n\nQuali comode funzionalità offre questo
programma?");

        System.out.print("\n\n-Gli URL da visitare possono essere passati anche tramite file
(workload.csv).");

        System.out.print("\n-Gli URL già visitati sono salvati in un file di testo e riletti
ogni volta che ");

        System.out.print("il programma viene avviato, per evitare di ripetere lavoro già fatto
in precedenza.");

```



```

        System.out.print("\n-Stessa cosa viene fatta per gli indirizzi email.");

        System.out.print("\n-I parametri di configurazione del programma sono contenuti in un
semplice file di ");

        System.out.print("testo e comprendono il numero massimo di thread crawlers che operano
in contemporanea ");

        System.out.print("e la profondità dell'analisi (espressa nel numero massimo di URL da
analizzare.");

        System.out.print("\n\n*-----*\n\nIl programma è coperto da copyright?");

        System.out.print("\n\nLa licenza con cui viene fornito questo software è la Attribution-
NonCommercial-ShareAlike ");

        System.out.print("3.0 Unported (CC BY-NC-SA 3.0). E' possibile modificare questo
software e ridistribuirlo ");

        System.out.print("solo riportandone l'autore originale (Giovanni Dini). Non è possibile
derivarne prodotti ");

        System.out.print("commerciali. Ogni derivazione dovrà essere distribuita sotto la stessa
licenza.\n");

        System.out.println("\nInserire l'URL da cui cominciare l'analisi, oppure digitare \"file
\" se ");

        System.out.println("si desidera leggere la workload dal file apposito ");

        System.out.println("(digitare \"help\" per istruzioni o \"exit\" per uscire:");

        first_url = user_input.next();

    }

    // System.out.println("Hai inserito questo testo: "+first_url);

/**

```

```

* Genero le liste che saranno utilizzate dal programma
*/

ArrayList<String> workload = new ArrayList<>();
ArrayList<String> visited = new ArrayList<>();
ArrayList<String> emails = new ArrayList<>();
ArrayList<Integer> param = new ArrayList<>();

// Se l'utente sceglie di uscire facciamo terminare il programma
if ("exit".equals(first_url)) {
    return;
}

// Se l'utente sceglie di operare a partire dai file, il programma segue
// queste istruzioni
File workloadFile = new File("workload.csv");
File lastWorkload = new File("nextWorkload.csv");
if ("file".equals(first_url)) {
    if (!workloadFile.exists()) {
        try {
            // Il file non esisteva, chiediamo all'utente di inserire
            // manualmente un URL

```

```

        workloadFile.createNewFile();

        System.out.println("\n\nIl file non esisteva. Aggiungere manualmente il primo
URL da analizzare: ");

        first_url = user_input.next();
    } catch (IOException ex) {
        //Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null, ex);
    }
} else {
    // System.out.println("Devo leggere il file.");
    Organizer.fileWorkload(workloadFile, workload);
    first_url = (String) workload.get(0);
    // System.out.println("Devo analizzare questo: "+first_url);
}
}

if ("restart".equals(first_url)) {
    if (!lastWorkload.exists()) {
        // Il file non esisteva, chiediamo all'utente di inserire
        // manualmente un URL

        System.out.println("\nIl file nextWorkload.csv non esiste. Aggiungere manualmente il
primo URL da analizzare:");

        first_url = user_input.next();
    } else {
        System.out.println("Riprendo il lavoro dall'ultima sessione.");
    }
}

```

```

        if (lastWorkload.length() == 0){
            System.out.println("Il file nextWorkload.csv era vuoto. Inserire manualmente un
URL:");

            first_url = user_input.next();
        } else {
            Organizer.fileWorkload(lastWorkload, workload);
            first_url = (String) workload.get(0);
        }
        //System.out.println("Devo analizzare questo: " + first_url);
    }
}

/**
 * Queste sono le istruzioni che il programma dovrà usare per operare
 * sui file che contengono email già trovate e indirizzi precedentemente
 * visitati, in modo da non fare più volte lo stesso lavoro.
 */

File emailsFile = new File("emails.csv");
File visitedFile = new File("visited.csv");

// Se l'utente sceglie di resettare il programma, cancelliamo i file che

```

```

// contengono le history delle precedenti sessioni. Se non esistono, li creiamo
// da zero.
if ("reset".equals(first_url)) {
    // Resettiamo il file che contiene le workload salvate dalle precedenti sessioni
    if (!lastWorkload.exists()) {
        try {
            lastWorkload.createNewFile();
        } catch (IOException ex) {
            ex);
            ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,
            }
            System.out.println("Il file nextWorkload.csv è stato resettato.");
        } else {
            lastWorkload.delete();
            try {
                lastWorkload.createNewFile();
            } catch (IOException ex) {
                ex);
                ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,
                }
            }
            System.out.println("Il file nextWorkload.csv è stato resettato.");

```

```

// Resettiamo il file che contiene le email salvate dalle precedenti sessioni
if (!emailsFile.exists()) {
    try {
        emailsFile.createNewFile();
    } catch (IOException ex) {
        ex);
        ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,

    }
    System.out.println("Il file emails.csv è stato resettato.");
} else {
    emailsFile.delete();
    try {
        emailsFile.createNewFile();
    } catch (IOException ex) {
        ex);
        ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,

    }
    System.out.println("Il file emails.csv è stato resettato.");
}

```

```

// Resettiamo il file che contiene gli URL visitati nelle precedenti sessioni
if (!visitedFile.exists()) {
    try {

```

```

        visitedFile.createNewFile();
    } catch (IOException ex) {
        ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,
ex);

    }
    System.out.println("Il file visited.csv è stato resettato.");
} else {
    visitedFile.delete();
    try {
        visitedFile.createNewFile();
    } catch (IOException ex) {
        ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null,
ex);

    }
    System.out.println("Il file visited.csv è stato resettato.");
}

System.out.println("\nOperazioni disponibili:");
System.out.println("- Inserire l'URL da cui cominciare l'analisi o \"exit\" per
uscire:");
user_input = new Scanner(System.in);
first_url = user_input.next();

```

```

        if ("exit".equals(first_url)) {
            return;
        }
    }

    /**
     * Creiamo il file per gestire le email trovate, se non esiste.
     */

    if (!emailsFile.exists()) {
        try {
            emailsFile.createNewFile();
        } catch (IOException ex) {
            ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Creiamo il file per gestire gli URL visitati, se non esiste.
     */

    if (!visitedFile.exists()) {

```



```

    try {
        visitedFile.createNewFile();
    } catch (IOException ex) {
        ////Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null, ex);
    }
}

// Aggiungiamo i dati delle precedenti sessioni nelle nostre liste.
Organizer.fileEmails(emailsFile, emails);
Organizer.fileVisited(visitedFile, visited);

// Aggiungiamo il primo URL e lo passiamo al nostro crawler.
workload.add(first_url);

/**
 * Genero il manager.
 */

// Inizializziamo le variabili che configureranno il nostro manager
int numCrawlers; // Indica il numero massimo di crawler che operano in contemporanea
int maxURLS; // Indica la profondità dell'analisi (numero di URL da visitare)

```

```

/**
 * Andiamo a leggere il file di configurazione che contiene i parametri
 * per configurare il manager.
 */

File paramFile = new File("preferences.cfg");
// Se il file di configurazione non esiste

if (!paramFile.exists()) {
    System.out.println("Il file di configurazione non esiste. Saranno utilizzati i parametri
di default: ");
    System.out.println("Numero di crawlers in contemporanea: 5.\nProfondità dell'analisi: 30
URLs.");
    numCrawlers = 5;
    maxURLS = 30;
} else {
    Organizer.fileParam(paramFile, param);
    // Se il file di configurazione è scritto male
    if (param.size() > 2) {
        System.out.println("Il file di configurazione non era scritto correttamente.");
        System.out.println("Controllare la documentazione per aiuto.");
        System.out.println("Saranno utilizzati i parametri di default: ");
    }
}

```

```

        System.out.println("Numero di crawlers in contemporanea: 5.\nProfondità
dell'analisi: 30 URLs.");
        numCrawlers = 5;
        maxURLS = 30;
    } else {
        // Se il file di configurazione è scritto bene, procediamo.
        numCrawlers = (Integer) param.get(0);
        maxURLS = (Integer) param.get(1);
        System.out.println("\nTrovato file di configurazione. Il programma avrà questi
parametri: ");
        System.out.println("-Numero di crawlers in contemporanea: " + numCrawlers);
        System.out.println("-Profondità dell'analisi: " + maxURLS + " URLs\n");
    }
}

/**
 * Generiamo il manager.
 */

Runnable manager = new Manager(workload, visited, emails, numCrawlers, maxURLS);
Thread m = new Thread(manager);

m.start();

```

```

m.setName("Manager");
try {
    m.join();
} catch (InterruptedException ex) {
    //Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null, ex);
}

/**
 * Terminato il manager, generiamo l'organizer per gestire l'output su
 * file.
 */

Runnable organizer = new Organizer(visited, emails);
Thread o = new Thread(organizer);

o.start();

o.setName("Organizer");
try {
    o.join();
} catch (InterruptedException ex) {

```

```

        //Logger.getLogger(ProgettoIngSoft.class.getName()).log(Level.SEVERE, null, ex);
    }

    System.out.println("L'analisi è stata completata. ");
    System.out.println("Ci sono ancora " + workload.size() + " URL nella workload.");
    System.out.println("Vuoi che siano scritti in un file in modo da poter proseguire l'analisi
    successivamente? si/no");
    String answer = user_input.next();

    if ("si".equals(answer)) {
        File finalWorkload = new File("nextWorkload.csv");
        Organizer.finalWorkload(finalWorkload, workload);
    }

    /**
     * Termino il thread main.
     */

    /**
     * Output di controllo
     *
     * if(m.isAlive() == true){ System.out.println("*-----*");

```

```

    * System.out.println("Il manager è vivo"); } else {
    * System.out.println("*-----*"); System.out.println("Il manager è
    * morto"); }
    *
    * if(o.isAlive() == true){ System.out.println("*-----*");
    * System.out.println("L'organizer è vivo"); } else {
    * System.out.println("*-----*"); System.out.println("L'organizer è
    * morto"); }
    *
    * System.out.println("*-----*"); System.out.println("Lista degli URL
    * visitati: "); Manager.printList(visited);
    * System.out.println("*-----*"); System.out.println("Lista delle
    * email trovate: "); Manager.printList(emails);
    * System.out.println("\n\nMain in attesa di terminazione.");
    *
    */
}

```

MANAGER.JAVA

```
package progettoingsoft;

import java.util.ArrayList;
import java.util.concurrent.Semaphore;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Manager implements Runnable {

    static ArrayList<String> workload;
    static ArrayList<String> visited;
    static ArrayList<String> emails;
    static int numCrawlers;
    static int maxURLS;
    static Semaphore stopGenerator;

    public Manager(ArrayList<String> workload, ArrayList<String> visited, ArrayList<String> emails,
int numCrawlers, int maxURLS) {
        Manager.workload = workload;
```

```

    Manager.visited = visited;
    Manager.emails = emails;
    Manager.numCrawlers = numCrawlers;
    Manager.maxURLS = maxURLS;
    // Questo semaforo limita il numero massimo di crawler che lavorano in contemporanea
    Manager.stopGenerator = new Semaphore(numCrawlers);
}

```

```

@Override

```

```

public void run() {

```

```

    /**
     * Il thread manager nasce.
     */
    // Stampe di controllo.
    // Thread m = Thread.currentThread();
    // String name = m.getName();
    // System.out.println("Thread "+name+" appena nato.");
    // System.out.println("Thread iniziali attivi: "+Thread.activeCount());

    /**
     * Genero il primo crawler, partendo dal primo URL.

```



```

*/

Runnable e = new Crawler(prendiURL(workload));
Thread t = new Thread(e);
getToken(); // Un token viene assegnato al crawler. Lo rilascerà in autonomia.
t.start();
t.setName("Crawler_start");
try {
    t.join();
} catch (InterruptedException ex) {
    Logger.getLogger(Manager.class.getName()).log(Level.SEVERE, null,
        ex);
}

/**
 * Genero nuovi crawlers. Partendo dai nuovi URL nella workload.
 */

int i = 1;
int analyzedURLS = 1;
// Se la lista degli URL da analizzare è vuota.
if (workload.isEmpty()) {

```

```

        System.out.println("\nATTENZIONE:");
        System.out.println("Non ci sono URL da analizzare.");
        System.out.println("Questo può essere dovuto alle seguenti cause: ");
        System.out.println("- L'URL iniziale (o il comando) non era ben scritto. Si ricorda di
usare la forma completa http(s)://www.sito.com");
        System.out.println("- L'URL iniziale era ben scritto ma la pagina non conteneva altri
URL.");
        System.out.println("- Gli URL trovati nella pagina iniziale erano già contenuti nel file
visited.csv, quindi sono già stati analizzati in una precedente sessione.");
        System.out.println("\nSe si crede che questo sia dovuto ad un problema, cancellare il
file \"visited.csv\".");
        System.out.println("\n\nIl programma termina ora.\n");
    } else {
        // Fin quando non si è raggiunta la profondità massima dell'analisi.
        while (analyzedURLS != maxURLS) {
            if (stopGenerator.availablePermits() > 0 && workload.size() > 0) {
                Runnable crawler1 = new Crawler(prendiURL(workload));
                Thread t1 = new Thread(crawler1);
                getToken();
                t1.start();
                t1.setName("Crawler" + i);
                i++;
                // Stampa di controllo.
            }
        }
    }
}

```

```

        // System.out.println("Numero stopGenerator:
"+stopGenerator.availablePermits());
        System.out.println("URL analizzati: " + analyzedURLS + " su un massimo di: " +
maxURLS);
        analyzedURLS++;
        // Stampa di controllo.
        // System.out.println("Ci sono " + workload.size() + " link in workload.");
        // System.out.println("Ci sono " + visited.size() + " link in visited.");
        // System.out.println("Ci sono " + Thread.activeCount() + " thread attivi.");
        // printList(visited);
    }
}

/**
 * Quando i crawlers sono terminati, terminiamo il thread manager.
 */

while (stopGenerator.availablePermits() != numCrawlers) {
    // Attendo che tutti i thread abbiano rilasciato il token.
}

// Stampa di controllo

```

```

        // System.out.println("THREAD "+name+" in attesa di terminazione. ");
    }

    /**
     * Questo metodo serve per prendere il primo URL contenuto nell'ArrayList
     * workload, ovvero la lista che contiene gli URL da visitare.
     *
     * @param workload la lista con gli URL da visitare.
     * @return il primo URL nella lista di attesa.
     */

    public synchronized String prendiURL(ArrayList<String> workload) {
        String URL;
        // Retrieving values from list
        int i = 0;
        Object obj = (workload.get(i));
        URL = obj.toString();
        workload.remove(i);
        visited.add(URL);
        return URL;
    }

```

```

/**
 * Questo metodo di controllo serve per stampare il contenuto di una lista.
 * E' usato come strumento di debug.
 *
 * @param list è la lista da stampare.
 */

public static void printList(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        int j = i + 1;
        System.out.println("Elemento numero " + j + " : " + list.get(i));
    }
}

/**
 * Questo metodo serve per controllare se l'URL trovato dal crawler è già
 * stato analizzato o si trova già nella workload.
 *
 * @param URL è l'url trovato dal crawler.
 */

public static void checkURL(String URL) {

```

```

        /* Cerco se l'elemento è già presente nella lista visited */
        if (visited.contains(URL) || workload.contains(URL)) {
            // System.out.println(URL+" già visitato.");
        } else {
            addURL(URL);
        }
    }

}

/**
 * Questo metodo serve per controllare se l'email trovata dal crawler è già
 * stata inserita nella lista.
 *
 * @param email è la email trovata dal crawler.
 */

public static void checkEmail(String email) {
    if (emails.contains(email)) {
        // System.out.println(URL+" già visitato.");
    } else {
        addEmail(email);
    }
}
}

```

```

/**
 * Questo metodo serve per inserire l'URL trovato dal crawler nella lista
 * degli URL in attesa di analisi. Viene invocato se il controllo sull'URL è
 * risultato nella non presenza dell'URL nelle liste workload o visited.
 *
 * @param URL è l'URL da inserire.
 */

public static void addURL(String URL) {
    workload.add(URL);
}

/**
 * Questo metodo serve per inserire l'indirizzo email trovato dal crawler
 * nella lista delle emails trovate. Viene invocato se il controllo
 * sull'email è risultato nella non presenza dell'URL nella lista email.
 *
 * @param email è l'indirizzo email da inserire.
 */

public static void addEmail(String email) {

```

```

        emails.add(email);
    }

    /**
     * Questo metodo serve per prendere un token dal semaforo e limitare così il
     * numero di thread contemporanei che vengono generati.
     */

    public static void getToken() {
        try {
            stopGenerator.acquire();
            // Stampa di controllo.
            // System.out.println("Token acquisito da un crawler:
"+stopGenerator.availablePermits());
            // System.out.println("THREAD ATTIVI: "+Thread.activeCount());
        } catch (InterruptedException ex) {
            Logger.getLogger(Manager.class.getName()).log(Level.SEVERE, null,
                ex);
        }
    }

    /**

```



```
* Questo metodo serve per rilasciare un token dal semaforo e lasciare così
* il posto ad un altro thread che può essere avviato in background.
*/

public static void releaseToken() {
    stopGenerator.release();
    // Stampa di controllo.
    // System.out.println("Token rilasciato da un crawler: "+stopGenerator.availablePermits());
}
}
```

CRAWLER.JAVA:

```
package progettoingsoft;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jsoup.Connection;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class Crawler implements Runnable {

    String URL;
    String name;

    /**
     * Passiamo i parametri al crawler.
     */
}
```

```

*
* @param URL
*/

public Crawler(String URL) {
    this.URL = URL;
}

/**
 * Metodo run, per l'esecuzione del crawler.
 * @param URL è l'URL della pagina da analizzare.
 */

@Override
public void run() {

    /**
     * Il thread crawler nasce.
     */

    // Stampe di controllo
    // Thread t = Thread.currentThread();

```

```

// String name = t.getName();
// System.out.println();
// System.out.println("Thread "+name+" appena nato.");
try {
    /**
     * Ci connettiamo alla pagina per controllare lo status che ci
     * ritorna e il tipo di documento.
     */

    // Controlliamo che l'URL passato sia ben formato.
    String[] URLcheck = URL.split("//");
    if (!"http:".equals(URLcheck[0])) {
        System.out.println("URL malformato. Il crawler termina ora.");
        return;
    }
    Connection.Response response = Jsoup.connect(URL).timeout(0).execute();
    int statusCode = response.statusCode();
    String contentType = response.contentType();
    // Se la pagina è disponibile.
    if (statusCode == 200) {
        try {
            // Filtriamo il risultato dell'interrogazione sul tipo di contenuto.

```

```

// (escludiamo per esempio il tipo di codifica della pagina - es.: UTF-8)
String[] contentTypeClean = contentType.split(";");
contentType = contentTypeClean[0];
// Se non è una pagina contenente testo non ci interessa.
if ("text/html".equals(contentType)
    || "text/plain".equals(contentType)
    || "text/css".equals(contentType)
    || "text/asp".equals(contentType)
    || "text/xml".equals(contentType)
    || "text/uri-list".equals(contentType)
    || "text/richtext".equals(contentType)) {
    Document doc = Jsoup.connect(URL).timeout(0).get();
    parsing(doc);
    // Stampa di controllo.
    // System.out.println(name+" "+statusCode+" "+contentType);
} else {
    // Stampa di controllo.
    // System.out.println("La pagina "+URL+" non è stata processata perchè aveva
questo formato: "+contentType);
}
} catch (IOException ex) {
    //Logger.getLogger(Crawler.class.getName()).log(Level.SEVERE,null, ex);
}

```

```

        }
    } else {
        // Stampa di controllo.
        // System.out.println("La pagina "+URL+" ha dato questo errore: "+statusCode);
    }
} catch (IOException ex) {
    // Stampa di controllo.
    // Logger.getLogger(Crawler.class.getName()).log(Level.SEVERE, null, ex);
    // System.out.println("404 ERROR");
}

// Stampa di controllo.
// System.out.print("Il thread "+name);

/**
 * Rilascio il token precedentemente acquisito.
 */

Manager.releaseToken();

/**
 * Il thread crawler termina.

```

```

        */

        // System.out.println("Thread "+name+" in fase di terminazione.");
        // System.out.println("Thread attivi prima di questa terminazione: "+Thread.activeCount());
    }

    /**
     * Questo metodo effettua il parsing della pagina.
     *
     * @param doc è il documento passato come risultato della connessione
     * all'URL.
     */

    protected void parsing(Document doc) {
        // Stampa di controllo.
        // System.out.println(" deve lavorare su questo indirizzo: "+URL);
        // Document doc = Jsoup.connect(URL).timeout(0).get();
        // System.out.println(doc);
        findLinksEmails(doc);
    }

    /**

```

```

* Questo metodo effettua la ricerca di links ed indirizzi email nella
* pagina parsata.
*
* @param doc è il documento passato come risultato del parsing.
*/

```

```

protected void findLinksEmails(Document doc) {
    Elements links = doc.select("a[href]");
    // System.out.println("\nLinks: (%d)", links.size());
    for (Element link : links) {
        // print(" * a: <%s> (%s)", link.attr("abs:href"),
        // trim(link.text(), 35));
        String content = link.attr("abs:href").toString();
        String emailPre = "mailto:";
        if (content.contains(emailPre)) {
            String newStr;
            newStr = link.attr("abs:href").replace(emailPre, "");
            newStr = newStr.replaceAll(" ", "");
            String[] newStrClean = newStr.split("\\?");
            newStr = newStrClean[0];
            Manager.checkEmail(newStr);
            // System.out.println("Ho trovato questa email: "+newStr);
        }
    }
}

```



```

        } else {
            Manager.checkURL(link.attr("abs:href"));
        }
    }
}

/**
 * Metodi non più usati: Questi metodi erano usati durante lo sviluppo del
 * software. Nelle ultime versioni non sono più usati, ma sono stati
 * mantenuti per comodità.
 *
 * Questo metodo serve per aggiungere l'email trovata alla lista.
 *
 * @param doc è il documento parsato.
 *
 * protected void findEmails(Document doc) { Elements emails =
 * doc.select("a[href]"); print("\nEmails: (%d)", emails.size()); for
 * (Element email : emails) { Manager.addEmail(email.attr("")); } }
 *
 * Questo metodo era usato per abbreviare la stampa in fase di stesura del
 * codice.
 * @param msg

```

```

* @param args
*
* private static void print(String msg, Object... args) {
*   System.out.println(String.format(msg, args)); }
*
*
* Questa funzione serviva per manipolare gli URL.
*
* @param s
* @param width
* @return
*
* private static String trim(String s, int width) { if (s.length() >
* width){ return s.substring(0, width-1) + "."; } else { return s; } }
*/
}

```

ORGANIZER.JAVA

```
package progettoingsoft;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Organizer implements Runnable {

    static ArrayList<String> visited;
    static ArrayList<String> emails;
```

```

public Organizer(ArrayList<String> visited, ArrayList<String> emails) {
    Organizer.visited = Manager.visited;
    Organizer.emails = Manager.emails;
}

/**
 * Il metodo run scrive le liste ottenute sui relativi file.
 */

@Override
public void run() {
    try {
        FileOutputStream outputEmails = new FileOutputStream("emails.csv");
        try (PrintStream writeEmails = new PrintStream(outputEmails)) {
            for (int i = 0; i < emails.size(); i++) {
                writeEmails.print(emails.get(i));
                writeEmails.print(",");
            }
        }
        System.out.println("*--*--*--*--*");
        System.out.println("La lista delle emails trovate è nel file emails.csv.");
        FileOutputStream outputVisited = new FileOutputStream("visited.csv");
    }
}

```

```

        try (PrintStream writeVisited = new PrintStream(outputVisited)) {
            for (int i = 0; i < visited.size(); i++) {
                writeVisited.print(visited.get(i));
                writeVisited.print(",");
            }
        }
        System.out.println("*--*--*--*--*");
        System.out.println("La lista degli URL visitati è nel file visited.csv.");
        System.out.println("*--*--*--*--*");
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
    }

/**
 * Stampe di controllo.
 *
 * System.out.println("*--*--*--*--*"); System.out.println("Lista degli
 * URL visitati:"); Manager.printList(visited);
 * System.out.println("*--*--*--*--*"); System.out.println("Lista degli
 * indirizzi email trovati:"); Manager.printList(emails);
 * System.out.println("*--*--*--*--*");
 */

```

```

}

/**
 * Questi metodi servono per importare i dati (URL da visitare, indirizzi
 * email già trovati, URL già visitati) da file.
 */

/**
 * Questo metodo importa i parametri del programma.
 *
 * @param file è il file su cui operiamo.
 * @param workload è la lista su cui operiamo
 * @return
 */

public static ArrayList<Integer> fileParam(File file, ArrayList<Integer> param) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;
        try {
            while ((line = br.readLine()) != null) {
                String[] value = line.split(",");

```

```

        for (String temp : value) {
            //System.out.println(email);
            int num = Integer.parseInt(temp);
            param.add(num);
        }
    }
} catch (IOException ex) {
    Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
}
} catch (FileNotFoundException ex) {
    Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
}
return param;
}

/**
 * Questo metodo importa gli URL da visitare nella workload.
 *
 * @param file è il file su cui operiamo.
 * @param workload è il file su cui operiamo.
 * @return
 */

```

```

public static ArrayList<String> fileWorkload(File file, ArrayList<String> workload) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;

        try {
            while ((line = br.readLine()) != null) {
                String[] value = line.split(",");
                for (String URL : value) {
                    //System.out.println(email);
                    workload.add(URL);
                }
            }
        } catch (IOException ex) {
            Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
        }

    } catch (FileNotFoundException ex) {
        Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
    }

    return workload;
}

```



```

}

/**
 * Questo metodo importa gli indirizzi email già trovati nella lista emails.
 *
 * @param file è il file su cui operare.
 * @param emails è la lista su cui operiamo.
 * @return
 */

public static ArrayList<String> fileEmails(File file, ArrayList<String> emails) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;

        try {
            while ((line = br.readLine()) != null) {
                String[] value = line.split(",");
                for (String email : value) {
                    //System.out.println(email);
                    emails.add(email);
                }
            }
        }
    }
}

```

```

        }
    } catch (IOException ex) {
        Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
    }

    } catch (FileNotFoundException ex) {
        Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
    }
    return emails;
}

/**
 * Questo metodo importa gli URL già visitati nella lista visited.
 *
 * @param file è il file su cui operare.
 * @param visited è la lista su cui operare.
 * @return
 */

public static ArrayList<String> fileVisited(File file, ArrayList<String> visited) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(file));
    }
}

```

```

String line;

try {
    while ((line = br.readLine()) != null) {
        String[] value = line.split(",");
        for (String URL : value) {
            //System.out.println(URL);
            visited.add(URL);
        }
    }
} catch (IOException ex) {
    Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
}

} catch (FileNotFoundException ex) {
    Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
}

return visited;
}

/**
 * Questo metodo serve per salvare il contenuto della workload, in modo che

```

```

    * sia riutilizzabile per una prossima sessione.
    *
    * @param file è il file su cui operiamo.
    * @param workload è la lista su cui operiamo.
    */

    public static void finalWorkload(File file, ArrayList<String> workload) {
        try {
            file.createNewFile();
            FileOutputStream outputWorkload = new FileOutputStream(file);
            try (PrintStream writeWorkload = new PrintStream(outputWorkload)) {
                for (int i = 0; i < workload.size(); i++) {
                    writeWorkload.print(workload.get(i));
                    writeWorkload.print(",");
                }
            }
        } catch (IOException ex) {
            Logger.getLogger(Organizer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

PREFERENCES.CFG (ESEMPIO):

4 , 50

In questo caso il 4 rappresenta il numero massimo di crawlers che operano in contemporanea, mentre il 50 rappresenta la profondità dell'analisi (il numero di URLs da visitare).

Test

TEST WHITE BOX:

I test white box sono stati eseguiti e hanno avuto esito positivo. Particolare cura è stata riposta nel collaudo per copertura dei cammini di base: il programma è stato eseguito più volte, esplorando tutte le opzioni, per fare in modo che tutte le singole istruzioni fossero eseguite almeno una volta.

TEST BLACK BOX:

In rosso è indicato l'input, in nero l'output. Sequenza di test black box:

- Nessun file presente nella directory di lancio del programma (workload.csv, nextWorkload.csv, emails.csv, visited.csv, preferences.cfg). URL da analizzare inserito manualmente:

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini

Matricola: 232274

Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

<http://www.google.it>

Il file di configurazione non esiste. Saranno utilizzati i parametri di default:

Numero di crawlers in contemporanea: 5.

Profondità dell'analisi: 30 URLs.

URL analizzati: 1 su un massimo di: 30

URL analizzati: 2 su un massimo di: 30

URL analizzati: 3 su un massimo di: 30

URL analizzati: 4 su un massimo di: 30

URL analizzati: 5 su un massimo di: 30

URL analizzati: 6 su un massimo di: 30

URL analizzati: 7 su un massimo di: 30

URL analizzati: 8 su un massimo di: 30

URL analizzati: 9 su un massimo di: 30

URL analizzati: 10 su un massimo di: 30

URL analizzati: 11 su un massimo di: 30

URL analizzati: 12 su un massimo di: 30

URL analizzati: 13 su un massimo di: 30

URL analizzati: 14 su un massimo di: 30

URL analizzati: 15 su un massimo di: 30

URL analizzati: 16 su un massimo di: 30

URL analizzati: 17 su un massimo di: 30

URL analizzati: 18 su un massimo di: 30

URL analizzati: 19 su un massimo di: 30

URL analizzati: 20 su un massimo di: 30

URL analizzati: 21 su un massimo di: 30

URL analizzati: 22 su un massimo di: 30

URL analizzati: 23 su un massimo di: 30

URL analizzati: 24 su un massimo di: 30

URL analizzati: 25 su un massimo di: 30

```

URL analizzati: 26 su un massimo di: 30
URL analizzati: 27 su un massimo di: 30
URL analizzati: 28 su un massimo di: 30
URL analizzati: 29 su un massimo di: 30
*_*_*_*_*_*_*_*_*_*
La lista delle emails trovate è nel file emails.csv.
*_*_*_*_*_*_*_*_*_*
La lista degli URL visitati è nel file visited.csv.
*_*_*_*_*_*_*_*_*_*
L'analisi è stata completata.
Ci sono ancora 638 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi
successivamente? si/no

```

si

```

gioggi2002@MacBook-Pro-di-Giovanni1 ~/N/P/dist$ java -jar ProgettoingSoft.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
http://www.google.it
Il file di configurazione non esiste. Saranno utilizzati i parametri di default:
Numero di crawler in contemporanea: 5.
Profondità dell'analisi: 30 URL.
URL analizzati: 1 su un massimo di: 30
URL analizzati: 2 su un massimo di: 30
URL analizzati: 3 su un massimo di: 30
URL analizzati: 4 su un massimo di: 30
URL analizzati: 5 su un massimo di: 30
URL analizzati: 6 su un massimo di: 30
URL analizzati: 7 su un massimo di: 30
URL analizzati: 8 su un massimo di: 30
URL analizzati: 9 su un massimo di: 30
URL analizzati: 10 su un massimo di: 30
URL analizzati: 11 su un massimo di: 30
URL analizzati: 12 su un massimo di: 30
URL analizzati: 13 su un massimo di: 30
URL analizzati: 14 su un massimo di: 30
URL analizzati: 15 su un massimo di: 30
URL analizzati: 16 su un massimo di: 30
URL analizzati: 17 su un massimo di: 30
URL analizzati: 18 su un massimo di: 30
URL analizzati: 19 su un massimo di: 30
URL analizzati: 20 su un massimo di: 30
URL analizzati: 21 su un massimo di: 30
URL analizzati: 22 su un massimo di: 30
URL analizzati: 23 su un massimo di: 30
URL analizzati: 24 su un massimo di: 30
URL analizzati: 25 su un massimo di: 30
URL analizzati: 26 su un massimo di: 30
URL analizzati: 27 su un massimo di: 30
URL analizzati: 28 su un massimo di: 30
URL analizzati: 29 su un massimo di: 30
*_*_*_*_*_*_*_*_*_*
La lista delle emails trovate è nel file emails.csv.
*_*_*_*_*_*_*_*_*_*
La lista degli URL visitati è nel file visited.csv.
*_*_*_*_*_*_*_*_*_*
L'analisi è stata completata.
Ci sono ancora 638 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi
successivamente? si/no
si

```

Al termine dell'esecuzione erano presenti i file emails.csv, visited.csv e nextWorkload.csv.

Dopo un controllo dei file, si è ritenuto che l'esecuzione fosse corretta.

- Ripresa della sessione precedente. Nessun file di configurazione presente.

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini
 Matricola: 232274
 Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

restart

Riprendo il lavoro dall'ultima sessione.

Il file di configurazione non esiste. Saranno utilizzati i parametri di default:

Numero di crawlers in contemporanea: 5.

Profondità dell'analisi: 30 URLs.

URL analizzati: 1 su un massimo di: 30

URL analizzati: 2 su un massimo di: 30

URL analizzati: 3 su un massimo di: 30

URL analizzati: 4 su un massimo di: 30

URL analizzati: 5 su un massimo di: 30

URL analizzati: 6 su un massimo di: 30

URL analizzati: 7 su un massimo di: 30

URL analizzati: 8 su un massimo di: 30

URL analizzati: 9 su un massimo di: 30

URL analizzati: 10 su un massimo di: 30

URL analizzati: 11 su un massimo di: 30

URL analizzati: 12 su un massimo di: 30

URL analizzati: 13 su un massimo di: 30

URL analizzati: 14 su un massimo di: 30

URL analizzati: 15 su un massimo di: 30

URL analizzati: 16 su un massimo di: 30

URL analizzati: 17 su un massimo di: 30

URL analizzati: 18 su un massimo di: 30

URL analizzati: 19 su un massimo di: 30

URL analizzati: 20 su un massimo di: 30

URL analizzati: 21 su un massimo di: 30

URL analizzati: 22 su un massimo di: 30

URL analizzati: 23 su un massimo di: 30

URL analizzati: 24 su un massimo di: 30

URL analizzati: 25 su un massimo di: 30

URL analizzati: 26 su un massimo di: 30

URL analizzati: 27 su un massimo di: 30

URL analizzati: 28 su un massimo di: 30

URL analizzati: 29 su un massimo di: 30

*_*_*_*_*_*_*_*_*_*

La lista delle emails trovate è nel file emails.csv.

*_*_*_*_*_*_*_*_*_*

La lista degli URL visitati è nel file visited.csv.

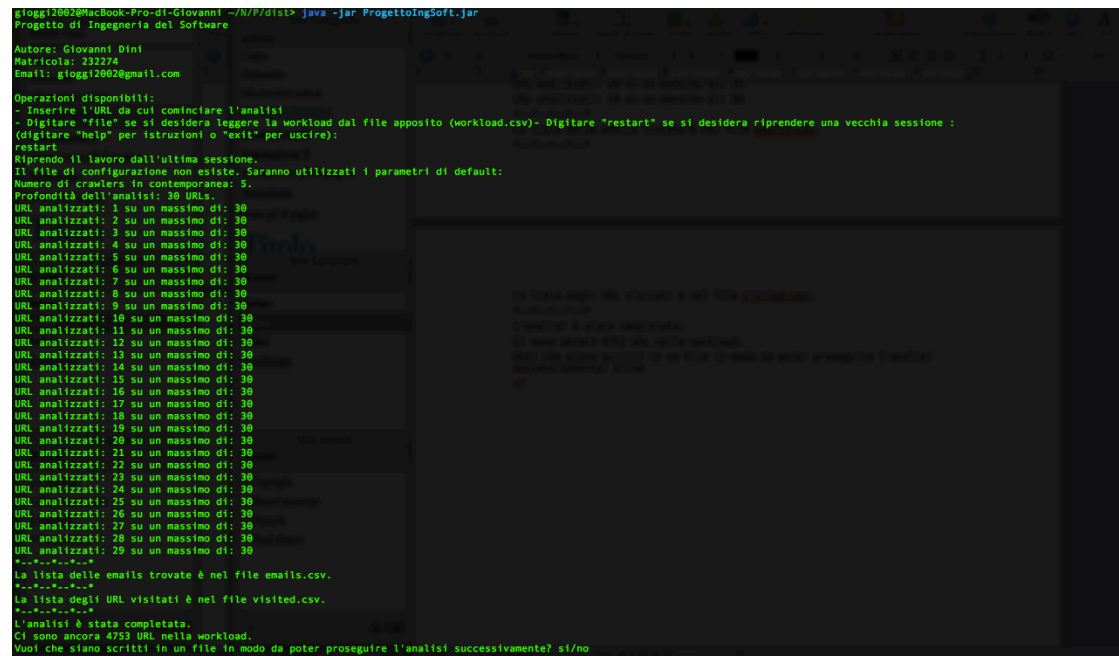
----*--*--*

L'analisi è stata completata.

Ci sono ancora 4753 URL nella workload.

Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no

si



```
gioggi2002@MacBook-Pro-di-Giovanni:~/P/disti$ java -jar ProgettoIngegneriaDelSoftware.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
restart
Riprendo il lavoro dall'ultima sessione.
Il file di configurazione non esiste. Saranno utilizzati i parametri di default:
Numero di crawlers in contemporanea: 5.
Profondità dell'analisi: 30 URLs.
URL analizzati: 1 su un massimo di: 30
URL analizzati: 2 su un massimo di: 30
URL analizzati: 3 su un massimo di: 30
URL analizzati: 4 su un massimo di: 30
URL analizzati: 5 su un massimo di: 30
URL analizzati: 6 su un massimo di: 30
URL analizzati: 7 su un massimo di: 30
URL analizzati: 8 su un massimo di: 30
URL analizzati: 9 su un massimo di: 30
URL analizzati: 10 su un massimo di: 30
URL analizzati: 11 su un massimo di: 30
URL analizzati: 12 su un massimo di: 30
URL analizzati: 13 su un massimo di: 30
URL analizzati: 14 su un massimo di: 30
URL analizzati: 15 su un massimo di: 30
URL analizzati: 16 su un massimo di: 30
URL analizzati: 17 su un massimo di: 30
URL analizzati: 18 su un massimo di: 30
URL analizzati: 19 su un massimo di: 30
URL analizzati: 20 su un massimo di: 30
URL analizzati: 21 su un massimo di: 30
URL analizzati: 22 su un massimo di: 30
URL analizzati: 23 su un massimo di: 30
URL analizzati: 24 su un massimo di: 30
URL analizzati: 25 su un massimo di: 30
URL analizzati: 26 su un massimo di: 30
URL analizzati: 27 su un massimo di: 30
URL analizzati: 28 su un massimo di: 30
URL analizzati: 29 su un massimo di: 30
*--*--*--*--*
La lista delle emails trovate è nel file emails.csv.
*--*--*--*--*
La lista degli URL visitati è nel file visited.csv.
*--*--*--*--*
L'analisi è stata completata.
Ci sono ancora 4753 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no
```

Al termine dell'esecuzione erano presenti i files emails.csv, visited.csv e nextWorkload.csv.

Dopo un controllo dei file, si è ritenuto che l'esecuzione fosse corretta.

- Richiesta d'aiuto e input della workload tramite file.

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar  
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

help

Cosa è e a cosa serve questo software?

Questo software genera un sistema di crawler che girano per la rete, raccogliendo indirizzi email e salvandoli per un successivo utilizzo.

Come si utilizza?

Il funzionamento è molto semplice: è sufficiente inserire un URL (nel formato http://www.sito.com) e il programma lo analizzerà, raccogliendo automaticamente gli indirizzi dei link che troverà e analizzando successivamente anche quelli. Tutti gli indirizzi email trovati durante l'analisi sono salvati per un successivo utilizzo.

Quale è l'utilità pratica di questo software?

In questa versione il software si limita a raccogliere indirizzi email (si ricorda che lo spam non è una pratica legale), ma con poche e semplici modifiche è possibile fargli raccogliere e analizzare praticamente tutto ciò che può essere contenuto in una pagina web. Vista la natura di questo software (realizzato per un progetto d'esame), è da considerarsi un puro esercizio, ma che con poche modifiche può avere scopi decisamente più utili.

Quali comode funzionalità offre questo programma?

-Gli URL da visitare possono essere passati anche tramite file (workload.csv).

-Gli URL già visitati sono salvati in un file di testo e rilette ogni volta che il programma viene avviato, per evitare di ripetere lavoro già fatto in precedenza.

-Stessa cosa viene fatta per gli indirizzi email.

* _ _ _ _ _ *

La licenza con cui viene fornito questo software è la Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). E' possibile modificare questo software e ridistribuirlo solo riportandone l'autore originale (Giovanni Dini). Non è possibile derivarne prodotti commerciali. Ogni derivazione dovrà essere distribuita sotto la stessa licenza.

file

Numero di crawlers in contemporanea: 5.

Profondità dell'analisi: 30 URLs.

URL analizzati: 1 su un massimo di: 30

URL analizzati: 2 su un massimo di: 30

URL analizzati: 3 su un massimo di: 30

URL analizzati: 4 su un massimo di: 30

URL analizzati: 5 su un massimo di: 30

URL analizzati: 6 su un massimo di: 30

URL analizzati: 7 su un massimo di: 30

URL analizzati: 8 su un massimo di: 30

URL analizzati: 9 su un massimo di: 30

URL analizzati: 10 su un massimo di: 30

URL analizzati: 11 su un massimo di: 30

URI analizzati: 12 su un massimo di: 30

URL analizzati: 13 su un massimo di: 30

URL analizzati: 14 su un massimo di: 30

URI analizzati: 15 su un massimo di: 30

URL analizzati: 16 su un massimo di: 30

URL analizzati: 17 su un massimo di: 30

URL analizzati: 18 su un massimo di: 30

URL analizzati: 19 su un massimo di: 30

URI analizzati: 20 su un massimo di: 30

URL analizzati: 21 su un massimo di: 30

URL analizzati: 22 su un massimo di: 30

URI analizzati: 23 su un massimo di: 30

URL analizzati: 24 su un massimo di: 30

URI analizzati: 25 su un massimo di: 30

URI analizzati: 26 su un massimo di: 30

URL analizzati: 27 su un massimo di: 30

URL analizzati: 28 su un massimo di: 30

URL analizzati: 29 su un massimo di: 30

* _ _ * _ _ * _ _ * _ _ *

67

*_*_*_*_*_*_*_*_*_*

La lista degli URL visitati è nel file visited.csv.

*_*_*_*_*_*_*_*_*_*

L'analisi è stata completata.

Ci sono ancora 1322 URL nella workload.

Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no

si

```
glogg12002@MacBook-Pro-di-Giovanni ~/N/P/Projetto di Ingegneria del Software: java -jar ProgettoingSoft.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: glogg12002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
help

*_*_*_*_*_*_*_*_*_*

Cosa è e a cosa serve questo software?

Questo software genera un sistema di crawler che girano per la rete, raccogliendo indirizzi email e salvandoli per un successivo utilizzo.

*_*_*_*_*_*_*_*_*_*

Come si utilizza?

Il funzionamento è molto semplice: è sufficiente inserire un URL (nel formato http://www.sito.com) e il programma lo analizzerà, raccogliendo automaticamente gli indirizzi dei link che troverà e analizzando successivamente anche quelli. Tutti gli indirizzi email trovati durante l'analisi sono salvati per un successivo utilizzo.

*_*_*_*_*_*_*_*_*_*

Quale è l'utilità pratica di questo software?

In questa versione il software si limita a raccogliere indirizzi email (si ricorda che lo spam non è una pratica legale), ma con poche e semplici modifiche è possibile fargli raccogliere e analizzare praticamente tutto ciò che può essere contenuto in una pagina web. Vista la natura di questo software (realizzato per un progetto d'esame), è da considerarsi un puro esercizio, ma che con poche modifiche può avere scopi decisamente più utili.

*_*_*_*_*_*_*_*_*_*

Quali comode funzionalità offre questo programma?

-Gli URL da visitare possono essere passati anche tramite file (workload.csv).
-Gli URL già visitati sono salvati in un file di testo e rilette ogni volta che il programma viene avviato, per evitare di ripetere lavoro già fatto in precedenza.
-Stessa cosa viene fatta per gli indirizzi email.
-I parametri di configurazione del programma sono contenuti in un semplice file di testo e comprendono il numero massimo di thread crawler che operano in contemporanea e la profondità dell'analisi (espressa nel numero massimo di URL da analizzare).

*_*_*_*_*_*_*_*_*_*

Il programma è coperto da copyright?

La licenza con cui viene fornito questo software è la Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). E' possibile modificare questo software e ridistribuirlo solo riportandone l'autore originale (Giovanni Dini). Non è possibile deriverne prodotti commerciali. Ogni derivazione dovrà essere distribuita sotto la stessa licenza.

Inserire l'URL da cui cominciare l'analisi, oppure digitare "file" se
```

```
*_*_*_*_*_*_*_*_*_*

Il programma è coperto da copyright?

La licenza con cui viene fornito questo software è la Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). E' possibile modificare questo software e ridistribuirlo solo riportandone l'autore originale (Giovanni Dini). Non è possibile deriverne prodotti commerciali. Ogni derivazione dovrà essere distribuita sotto la stessa licenza.

Inserire l'URL da cui cominciare l'analisi, oppure digitare "file" se
si desidera leggere la workload dal file apposito
(digitare "help" per istruzioni o "exit" per uscire):
file
Il file di configurazione non esiste. Saranno utilizzati i parametri di default:
Numero di crawlers in contemporanea: 5.
Profondità dell'analisi: 30 URL.
URL analizzati: 1 su un massimo di: 30
URL analizzati: 2 su un massimo di: 30
URL analizzati: 3 su un massimo di: 30
URL analizzati: 4 su un massimo di: 30
URL analizzati: 5 su un massimo di: 30
URL analizzati: 6 su un massimo di: 30
URL analizzati: 7 su un massimo di: 30
URL analizzati: 8 su un massimo di: 30
URL analizzati: 9 su un massimo di: 30
URL analizzati: 10 su un massimo di: 30
URL analizzati: 11 su un massimo di: 30
URL analizzati: 12 su un massimo di: 30
URL analizzati: 13 su un massimo di: 30
URL analizzati: 14 su un massimo di: 30
URL analizzati: 15 su un massimo di: 30
URL analizzati: 16 su un massimo di: 30
URL analizzati: 17 su un massimo di: 30
URL analizzati: 18 su un massimo di: 30
URL analizzati: 19 su un massimo di: 30
URL analizzati: 20 su un massimo di: 30
URL analizzati: 21 su un massimo di: 30
URL analizzati: 22 su un massimo di: 30
URL analizzati: 23 su un massimo di: 30
URL analizzati: 24 su un massimo di: 30
URL analizzati: 25 su un massimo di: 30
URL analizzati: 26 su un massimo di: 30
URL analizzati: 27 su un massimo di: 30
URL analizzati: 28 su un massimo di: 30
URL analizzati: 29 su un massimo di: 30
*_*_*_*_*_*_*_*_*_*
La lista delle emails trovate è nel file emails.csv.
*_*_*_*_*_*_*_*_*_*
La lista degli URL visitati è nel file visited.csv.
*_*_*_*_*_*_*_*_*_*
L'analisi è stata completata.
Ci sono ancora 1322 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no
si
```

Il programma risponde come previsto.

- Inserimento URL malformato.

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar  
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

ciao

Il file di configurazione non esiste. Saranno utilizzati i parametri di default:

Numero di crawlers in contemporanea: 5.

Profondità dell'analisi: 30 URLs.

URL malformato. Il programma termina ora.

ATTENZIONE:

Non ci sono URL da analizzare.

Questo può essere dovuto alle seguenti cause:

- L'URL iniziale (o il comando) non era ben scritto. Si ricorda di usare la forma completa http(s)://www.sito.com
- L'URL iniziale era ben scritto ma la pagina non conteneva altri URL.
- Gli URL trovati nella pagina iniziale erano già contenuti nel file visited.csv, quindi sono già stati analizzati in una precedente sessione.

Se si crede che questo sia dovuto ad un problema, cancellare il file "visited.csv".

Il programma termina ora.

*_*_*_*_*_*_*_*_*_*

La lista delle emails trovate è nel file emails.csv.

*_*_*_*_*_*_*_*_*_*

La lista degli URL visitati è nel file visited.csv.

*_*_*_*_*_*_*_*_*_*

L'analisi è stata completata.

Ci sono ancora 0 URL nella workload.

Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no

no

```
giogg12002@MacBook-Pro-di-Giovanni ~/H/F/dist> java -jar ProgettoingSoft.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: giogg12002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
ciao
Il file di configurazione non esiste. Saranno utilizzati i parametri di default:
Numero di Crawlers in contemporanea: 5.
Profondità dell'analisi: 30 URL.
URL malformato. Il programma termina ora.

ATTENZIONE:
Non ci sono URL da analizzare.
Questo può essere dovuto alle seguenti cause:
- L'URL iniziale (o il comando) non era ben scritto. Si ricorda di usare la forma completa http(s)://www.sito.com
- L'URL iniziale era ben scritto ma la pagina non conteneva altri URL.
- Gli URL trovati nella pagina iniziale erano già contenuti nel file visited.csv, quindi sono già stati analizzati in una precedente sessione.

Se si crede che questo sia dovuto ad un problema, cancellare il file "visited.csv".

Il programma termina ora.

+---+---+---+
La lista delle emails trovate è nel file emails.csv.
+---+---+---+
La lista degli URL visitati è nel file visited.csv.
+---+---+---+
L'analisi è stata completata.
Ci sono ancora 0 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no
no
giogg12002@MacBook-Pro-di-Giovanni ~/H/F/dist>
```

Come si può notare, il programma risponde adeguatamente.

- Esecuzione del programma quando nella sua directory è incluso un file preferences.cfg con il seguente contenuto:

3,10

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

<http://www.giovannidini.com>

Trovato file di configurazione. Il programma avrà questi parametri:

- Numero di crawlers in contemporanea: 3
- Profondità dell'analisi: 10 URLS

```
URL analizzati: 1 su un massimo di: 10
URL analizzati: 2 su un massimo di: 10
URL analizzati: 3 su un massimo di: 10
URL analizzati: 4 su un massimo di: 10
URL analizzati: 5 su un massimo di: 10
URL analizzati: 6 su un massimo di: 10
URL analizzati: 7 su un massimo di: 10
URL analizzati: 8 su un massimo di: 10
URL analizzati: 9 su un massimo di: 10
```

*_*_*_*_*_*_*_*_*_*

La lista delle emails trovate è nel file emails.csv.

*_*_*_*_*_*_*_*_*_*

La lista degli URL visitati è nel file visited.csv.

*_*_*_*_*_*_*_*_*_*

L'analisi è stata completata.

Ci sono ancora 223 URL nella workload.

Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no

no


```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> vim preferences.cfg
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
http://www.giovannidini.com

Trovato file di configurazione. Il programma avrà questi parametri:
-Numero di crawlers in contemporanea: 3
-Profondità dell'analisi: 10 URLS

URL analizzati: 1 su un massimo di: 10
URL analizzati: 2 su un massimo di: 10
URL analizzati: 3 su un massimo di: 10
URL analizzati: 4 su un massimo di: 10
URL analizzati: 5 su un massimo di: 10
URL analizzati: 6 su un massimo di: 10
URL analizzati: 7 su un massimo di: 10
URL analizzati: 8 su un massimo di: 10
URL analizzati: 9 su un massimo di: 10
+---+---+---+
La lista delle emails trovate è nel file emails.csv.
+---+---+---+
La lista degli URL visitati è nel file visited.csv.
+---+---+---+
L'analisi è stata completata.
Ci sono ancora 223 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no
no
```

Il programma procede come previsto.

- L'utente chiede di uscire.

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar  
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini

Matricola: 232274

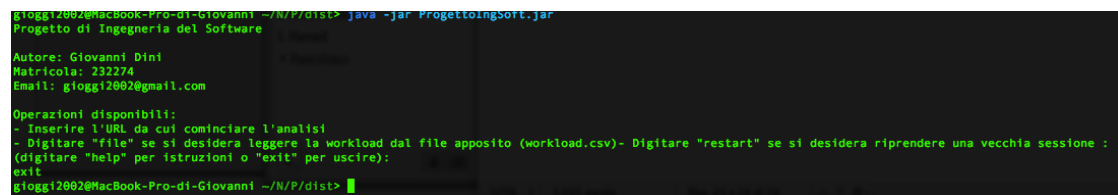
Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

exit



```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar  
Progetto di Ingegneria del Software  
  
Autore: Giovanni Dini  
Matricola: 232274  
Email: gioggi2002@gmail.com  
  
Operazioni disponibili:  
- Inserire l'URL da cui cominciare l'analisi  
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :  
(digitare "help" per istruzioni o "exit" per uscire):  
exit  
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist>
```

Il programma è eseguito correttamente.

- Il file nextWorkload.csv esiste ma è vuoto e l'utente prova a riprendere una sessione precedente.

```
gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar  
Progetto di Ingegneria del Software
```

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:

- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :

(digitare "help" per istruzioni o "exit" per uscire):

restart

Riprendo il lavoro dall'ultima sessione.

Il file nextWorkload.csv era vuoto. Inserire manualmente un URL:

<http://www.giovannidini.com>

Trovato file di configurazione. Il programma avrà questi parametri:

- Numero di crawlers in contemporanea: 4
- Profondità dell'analisi: 10 URLS

URL analizzati: 1 su un massimo di: 10
URL analizzati: 2 su un massimo di: 10
URL analizzati: 3 su un massimo di: 10
URL analizzati: 4 su un massimo di: 10
URL analizzati: 5 su un massimo di: 10
URL analizzati: 6 su un massimo di: 10
URL analizzati: 7 su un massimo di: 10
URL analizzati: 8 su un massimo di: 10
URL analizzati: 9 su un massimo di: 10

*_*_*_*_*_*_*_*_*_*

La lista delle emails trovate è nel file emails.csv.

*_*_*_*_*_*_*_*_*_*

La lista degli URL visitati è nel file visited.csv.

*_*_*_*_*_*_*_*_*_*

L'analisi è stata completata.

Ci sono ancora 221 URL nella workload.

Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no

si

```

gioggi2002@MacBook-Pro-di-Giovanni ~/N/P/dist> java -jar ProgettoIngSoft.jar
Progetto di Ingegneria del Software

Autore: Giovanni Dini
Matricola: 232274
Email: gioggi2002@gmail.com

Operazioni disponibili:
- Inserire l'URL da cui cominciare l'analisi
- Digitare "file" se si desidera leggere la workload dal file apposito (workload.csv)- Digitare "restart" se si desidera riprendere una vecchia sessione :
(digitare "help" per istruzioni o "exit" per uscire):
restart
Riprendo il lavoro dall'ultima sessione.
Il file nextWorkload.csv era vuoto. Inserire manualmente un URL:
http://www.giovannidini.com

Trovato file di configurazione. Il programma avrà questi parametri:
-Numero di crawlers in contemporanea: 4
-Profondità dell'analisi: 10 URLS

URL analizzati: 1 su un massimo di: 10
URL analizzati: 2 su un massimo di: 10
URL analizzati: 3 su un massimo di: 10
URL analizzati: 4 su un massimo di: 10
URL analizzati: 5 su un massimo di: 10
URL analizzati: 6 su un massimo di: 10
URL analizzati: 7 su un massimo di: 10
URL analizzati: 8 su un massimo di: 10
URL analizzati: 9 su un massimo di: 10
+---+---+---+
La lista delle emails trovate è nel file emails.csv.
+---+---+---+
La lista degli URL visitati è nel file visited.csv.
+---+---+---+
L'analisi è stata completata.
Ci sono ancora 221 URL nella workload.
Vuoi che siano scritti in un file in modo da poter proseguire l'analisi successivamente? si/no
si

```

Compilazione ed esecuzione

- Il progetto è stato sviluppato su :

Mac OS X Versione 10.7.4 in esecuzione su x86_64; US-ASCII; it_IT (nb).

- L'ambiente di sviluppo utilizzato è :

NetBeans IDE 7.2 (Build 201207301726).

- La versione di Java utilizzata è

1.7.0_07; Java HotSpot(TM) 64-Bit Server VM 23.3-b01

- La libreria esterna utilizzata nel progetto è Jsoup nella sua versione 1.6.3 e può essere scaricata a questo indirizzo: <http://jsoup.org/download>.

- I file sorgenti del progetto sono contenuti nella cartella:

ProgettoIngSoft/src/progettoingsoft/

- La libreria Jsoup è contenuta nella cartella:

ProgettoIngSoft/lib/Jsoup

- Il progetto può essere eseguito, dopo essersi portati nella cartella che contiene il file .jar compilato, con il comando:

java -jar ProgettoIngSoft.jar