



UNIVERSITÀ DI PISA

LAUREA MAGISTRALE IN
INFORMATICA UMANISTICA

PROGRAMMAZIONE E ANALISI DI DATI
ESPERIENZE DI PROGRAMMAZIONE IN JAVA
A.A. 2019/20

Valentina Righetti
Matricola: 604013

Indice

Descrizione del progetto 3

MainClass 3

Analisi 4

Output 5

Risultati 6

Listato del codice..... 8

 MainClass 8

 Analisi 9

 Output 17

Descrizione del progetto

Il progetto realizzato ha lo scopo di scompattare le cartelle del file ISO del progetto Gutenberg – prendendo in considerazione le cartelle da 1 a 9 – trovare i file con estensione TXT (corrispondenti a libri digitalizzati), recuperare i soli testi in inglese e analizzarne le frequenze delle frequenze delle occorrenze presenti in ciascuno. Il file ISO di partenza, delle dimensioni di circa 8 GB, alla fine del processo ha fornito 19.401 file di testo adatti per l'analisi.

Il progetto si suddivide in tre classi:

- **MainClass**, che contiene il main e lancia il programma che estrae, analizza i dati estratti e genera i file CSV con i risultati;
- **Analisi**, che contiene i metodi di estrazione, analisi dei file nella cartella di destinazione, per ricavare i soli documenti di testo, e di calcolo delle frequenze e delle frequenze delle frequenze;
- **Output**, che scrive su due file CSV i risultati ottenuti. La decisione di scrivere su due file di tipo CSV è stata dettata dalla possibilità, tramite gli stessi, di ricavare un grafico finale e, di conseguenza, apprezzare meglio i risultati.

MainClass

La classe MainClass contiene il main e, dunque, è quella che avvia il programma.

Per prima cosa inizializza le tre stringhe dei percorsi del file ZIP, della cartella Unzip, nella quale saranno inseriti i file che verranno estratti, e della cartella Output, nella quale saranno inseriti i due file CSV con i risultati. Vengono poi anche inizializzate le quattro HashMap necessarie per la registrazione delle informazioni sui libri che permetteranno al programma di elaborare i risultati:

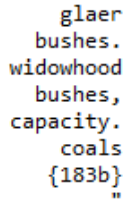
- **testi**, la cui chiave è un intero che rappresenta il numero univoco dell'e-book e il cui valore è la codifica necessaria per aprirlo. Contiene tutti i file con estensione TXT, i quali dovranno essere analizzati per tenere solo quelli che rispecchiano certe caratteristiche di lingua e codifica;
- **libri**, la cui chiave è un intero che rappresenta il numero univoco dell'e-book e il cui valore è il file stesso dell'e-book. Contiene tutti i file TXT che hanno passato l'analisi di lingua e codifica – e, dunque, sono in lingua inglese e con una codifica non nulla – e che possono essere utilizzati per calcolare frequenze e frequenze delle frequenze;
- **frequenze**, la cui chiave è una stringa che rappresenta una parola (un'occorrenza, cioè, dei file di testo) e il cui valore è un intero che rappresenta il numero di volte in cui tale parola appare in tutti i file analizzati;
- **frequenze_frequenze**, la cui chiave è un intero che rappresenta il numero di parole presenti lo stesso numero di volte (e quindi con la stessa frequenza) e il cui valore è un intero che rappresenta la frequenza stessa.

Infine, vengono inizializzati un oggetto della classe Analisi e uno della classe Output – per poterne invocare i metodi –, viene creata la directory Unzip (se già esistente si passa direttamente all'analisi dei file al suo interno) e viene lanciato il processo di estrazione e analisi dei file. Il programma avverte, con dei messaggi rivolti all'utente, quali azioni sta compiendo e se il processo è stato completato correttamente.

Analisi

La classe Analisi può essere considerata come il fulcro centrale del programma: al suo interno sono presenti i metodi che svolgono le azioni di estrazione e analisi nonché di calcolo delle frequenze e delle frequenze delle frequenze.

I metodi sono i seguenti:

- **Analisi del file ISO:**
 - **analisiISO:** analizza il file ISO di partenza estraendo i file ZIP contenuti nelle cartelle da 1 a 9. I documenti vengono analizzati ad uno ad uno: nel caso in cui il nome del file – trasformato interamente in minuscolo per evitare problemi di *case sensitivity*, dato che alcuni documenti hanno estensioni con le lettere maiuscole e altri con le lettere minuscole – abbia un'estensione “.zip” viene invocato il metodo `estraiZip` su di stesso; se si tratta, invece, di una cartella, viene richiamato il metodo `analisiISO` (quindi sé stesso) sul file;
 - **estraiZip:** se invocato su un file ZIP lo estrae e lo scrive nella cartella di destinazione `Unzip`; viene invocato solo su file che, con il metodo `analisiISO`, risultano avere una estensione “.zip”.
- **Analisi dei file di testo:**
 - **analisiFileTxt:** analizza i file contenuti nella cartella `Unzip` e, se di estensione “.txt” – di nuovo i nomi dei file vengono trasformati in lettere minuscole per evitare problemi di *case sensitivity* – vengono analizzati ricercandone la lingua (che deve essere inglese), la codifica (per poterli accedere in seguito con l'*encoding* corretto) e il numero identificativo univoco. Il controllo viene fatto dentro un ciclo `while` la cui guardia prevede che l'analisi prosegua finché il documento non è finito o finché non è stata raggiunta la riga col pattern “***START”, che indica la fine dei metadati del testo. Se il pattern è stato raggiunto e le informazioni (lingua, codifica e ID univoco) non sono state raccolte, il libro non viene preso in considerazione. Una volta trovate tutte le informazioni necessarie, se la lingua del libro è in inglese e la sua codifica non è nulla, il file di testo viene inserito nelle `HashMap` testi e libri. Essendo la chiave delle due `HashMap` il numero univoco del libro, nel caso in cui ci siano due copie dello stesso testo (con lo stesso ID), il nuovo file sostituisce il precedente; ci sarà, dunque, una sola copia per ogni libro.
- **Calcolo delle frequenze:**
 - **calcolaFrequenze:** partendo dalle `HashMap` testi e libri, calcola il numero di parole trovate e le inserisce nella `HashMap` frequenze. Il calcolo delle parole avviene per il solo testo effettivo del libro, escludendo tutte le informazioni e i metadati; per questo ci si avvale dell'utilizzo di due booleani, `testa` e `coda`, che indicano rispettivamente l'inizio e la fine del testo effettivo. Ogni file di testo (contenuto nella `HashMap` libri) viene acceduto con l'*encoding* corretto, salvato nella `HashMap` testi: questo avviene perché le due `HashMap` contengono gli stessi libri alle stesse posizioni e, dunque, la chiave, a parità di posizione, è la stessa e fa riferimento allo stesso libro.
Per trovare le parole si fa utilizzo di un `Pattern` che elimina punteggiatura, numeri e caratteri speciali. L'uso di `Pattern` in relazione a `Matcher` è stato previsto successivamente ad un output delle parole “scorretto”: contenevano anche, ad esempio, i segni di punteggiatura e gli spazi. L'immagine a destra raffigura un esempio delle parole rilevate dal programma prima dell'utilizzo di `Pattern`.
 - **calcolaFrequenzeDelleFrequenze:** analizza l'`HashMap` frequenze e calcola quante parole appaiono lo stesso numero di volte (e cioè con la stessa frequenza). I risultati del calcolo vengono inseriti nella `HashMap` `frequenze_frequenze`.

Output

La classe Output scrive i due file CSV con i risultati ottenuti dai calcoli effettuati dalla classe Analisi. È composta da due metodi che scrivono rispettivamente il file delle frequenze delle parole e quello delle frequenze delle frequenze:

- **scriviFileFrequenze:** crea la cartella Output (se non è già esistente) e il file dal nome “Frequenza_parole.csv” nel quale vengono scritti i risultati del calcolo delle frequenze.
Per la scrittura ci si avvale dell’uso di un FileWriter al quale è passato il file CSV appena creato; prima della scrittura effettiva, però, la HashMap frequenze viene ordinata grazie al *sorting* in *reverse order* fornito dall’interfaccia Comparator. In questo modo sarà più facile apprezzare i risultati dato che le parole con le frequenze più alte verranno poste per prime nella lista.
Il file CSV finale presenta due colonne: la prima contiene la parola e la seconda la frequenza con cui appare nei testi.
- **scriviFileFrequenzeFrequenze:** crea il file dal nome “Frequenza_frequenze.csv” nel quale vengono scritti i risultati del calcolo delle frequenze delle frequenze.
Di nuovo si è utilizzato un FileWriter per scrivere i dati sul file; anche in questo caso si è optato per un ordinamento in senso decrescente dell’HashMap frequenze_frequenze per apprezzare meglio i risultati.
Il file CSV presenta due colonne: la prima con le frequenze delle frequenze e la seconda con le frequenze stesse.

Risultati

Dei 26.095 file di testo (con estensione TXT) trovati dal programma, 19.401 erano in lingua inglese e con una codifica non nulla e, per questo, adatti all'analisi.

I risultati del calcolo delle frequenze – del numero di parole, cioè, presenti negli oltre 19.000 testi analizzati – hanno prodotto più di mezzo milione di parole diverse fra loro.

La tabella che riporta i calcoli delle frequenze delle parole conferma l'ipotesi secondo cui occorrenze come congiunzioni o pronomi siano le più utilizzate e, per questo, abbiano una maggior frequenza. Di seguito si riportano, in parte, i risultati ottenuti.

ANALISI DEI FILE DI TESTO:

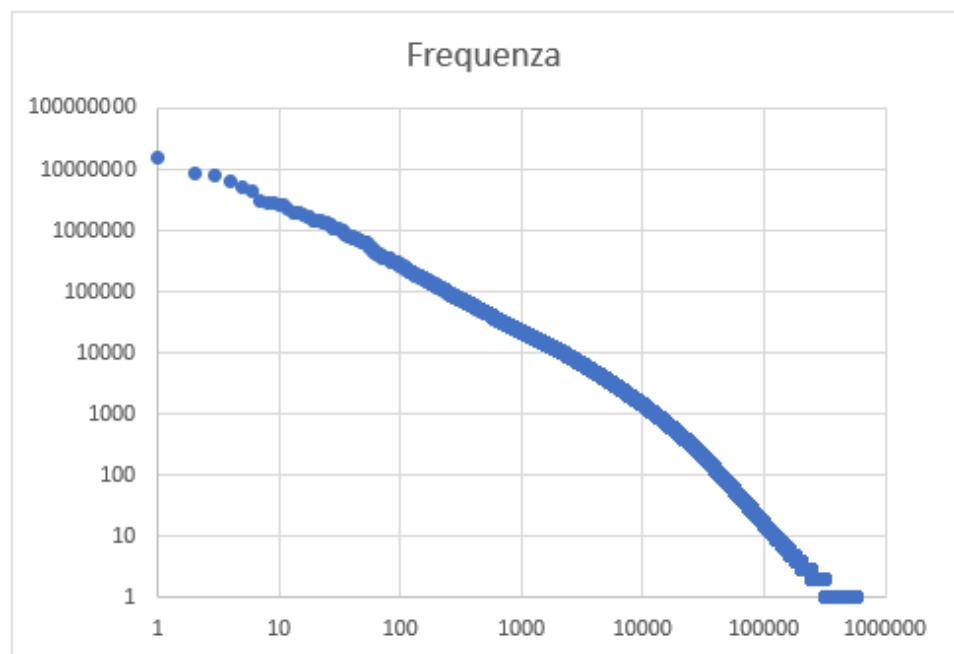
Numero totale di file txt: 26095

Numero di file txt in inglese da analizzare: 19401

CALCOLO DELLE FREQUENZE:

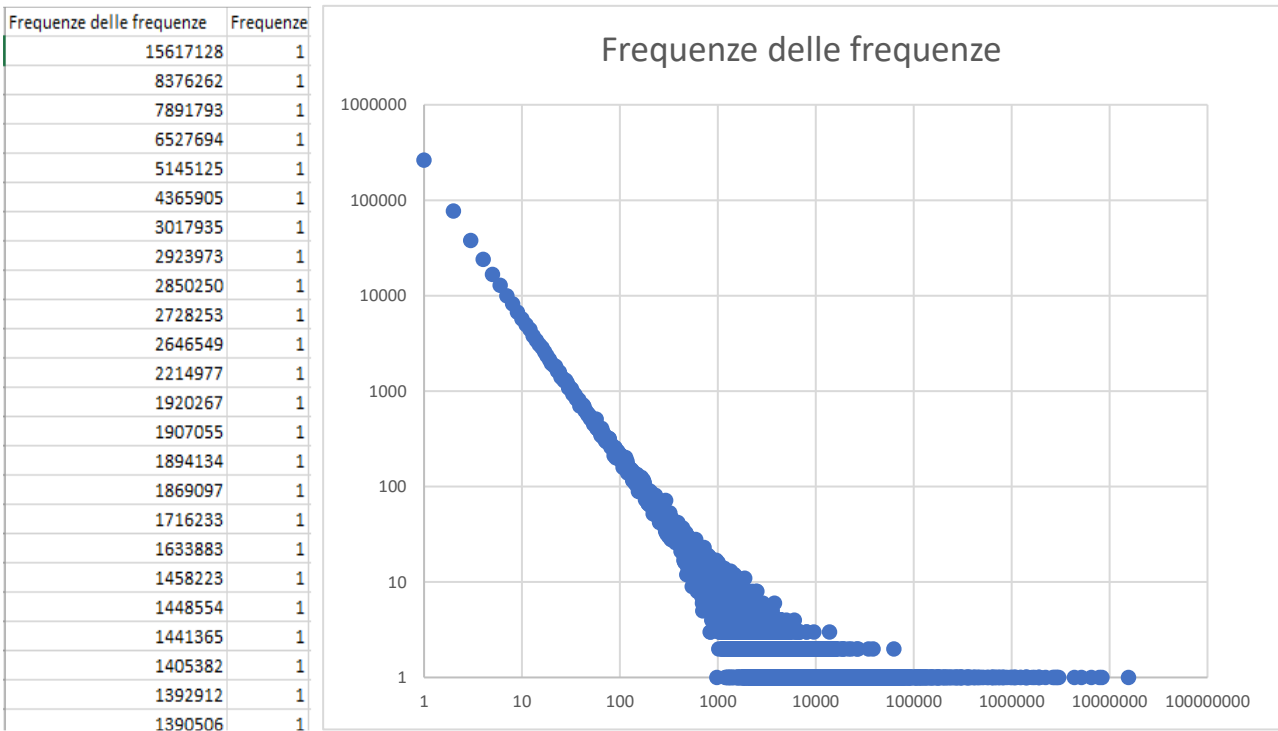
Numero totale di parole trovate: 584915

| Parola | Frequenza |
|--------|-----------|
| the | 15617128 |
| of | 8376262 |
| and | 7891793 |
| to | 6527694 |
| a | 5145125 |
| in | 4365905 |
| that | 3017935 |
| i | 2923973 |
| he | 2850250 |
| was | 2728253 |
| it | 2646549 |
| his | 2214977 |
| is | 1920267 |
| as | 1907055 |
| with | 1894134 |
| for | 1869097 |
| you | 1716233 |



Come si può notare, le parole con una frequenza maggiore sono proprio le congiunzioni, gli articoli e i pronomi personali. Anche il grafico mostra come queste parole ricorrano con una frequenza molto alta rispetto alla maggior parte degli altri vocaboli che, invece, hanno una frequenza molto più bassa. In generale, sono poche le parole che ricorrono con un'alta frequenza mentre la maggior parte è più "rara" o, comunque, meno comune e, di conseguenza, ha una frequenza molto più bassa.

Per quanto riguarda i calcoli delle frequenze delle frequenze, invece, i risultati sono stati i seguenti:



Anche in questo caso viene dimostrato che le parole con un’alta frequenza – che, dunque, occorrono molte volte nei testi analizzati – sono poche, mentre le parole con una bassa frequenza sono la maggior parte.

Listato del codice

MainClass

```
import java.io.*;
import java.util.*;
import java.nio.charset.Charset;

public class MainClass {
    public static void main(String[] args) {

        //file zip di partenza
        String file_zip = "\\Desktop\\iso";
        //directory in cui il file viene unzippato
        String dir_dest = "\\Desktop\\Unzip";
        //directory in cui vengono inseriti i calcoli delle frequenze
        String dir_risultato = "\\Desktop\\Output";

        //Integer e' l'ID univoco del libro, Charset e' la codifica
        HashMap<Integer, Charset> testi = new HashMap<>();
        //Integer e' l'ID univoco del libro, File e' il libro stesso
        HashMap<Integer, File> libri = new HashMap<>();
        //String e' la parola, Integer e' il numero di volte
        //che appare nei testi
        HashMap<String, Integer> frequenze = new HashMap<>();
        //la key Integer e' il numero di parole che compaiono
        //lo stesso numero di volte (frequenze),
        //il value Integer e' il numero di volte (frequenze delle
        // frequenze)
        HashMap<Integer, Integer> frequenze_frequenze = new HashMap<>();

        Analisi analisi = new Analisi();
        Output output = new Output();

        File directory = new File(dir_dest);
        //se la directory non esiste viene creata e vi vengono
        //inseriti i file estratti
        if (!directory.exists()) {
            //estrae i file dallo zip
            System.out.println("UNZIP DEL FILE ISO...");
            analisi.analisiISO(file_zip, dir_dest);
        }
    }
}
```



```

    }

    System.out.println();
    //analizza i file estratti
    System.out.println("ANALISI DEI FILE DI TESTO...");
    analisi.analisiFileTxt(directory, testi, libri);

    System.out.println();
    //calcola le frequenze delle parole e le frequenze
    //delle frequenze (quante parole appaiono lo stesso
    //numero di volte)
    System.out.println("CALCOLO DELLE FREQUENZE...");
    analisi.calcolaFrequenze(testi, libri, frequenze);
    analisi.calcolaFrequenzeDelleFrequenze(frequenze,
        frequenze_frequenze);

    System.out.println();
    //genera i due file CSV che contengono i risultati
    //dei calcoli delle frequenze
    System.out.println("GENERAZIONE DEI FILE CSV...");
    output.scriviFileFrequenze(dir_risultato, frequenze);
    output.scriviFileFrequenzeFrequenze(dir_risultato,
        frequenze_frequenze);

    System.out.println();
    System.out.println("OPERAZIONE COMPLETATA CON SUCCESSO");
    System.out.println();
    System.out.println("I file con i risultati si trovano nella
        cartella " + dir_risultato);
    }
}

```

Analisi

```

import java.io.*;
import java.nio.charset.*;
import java.util.*;
import java.util.regex.*;
import java.util.zip.*;

public class Analisi {

```

```

//metodo che analizza il file ISO e richiama il
//metodo di estrazione dei file ZIP da esso
public void analisiISO(String zip, String dir) {
    File path_zip = new File(zip);
    //crea un array che contiene tutti i file dello ZIP
    File[] lista_file_zip = path_zip.listFiles();

    if (lista_file_zip != null) {
        for (File file : lista_file_zip) {
            //il nome viene trasformato tutto in minuscolo
            //per evitare problemi di case sensitivity
            String nome = file.getName().toLowerCase();

            //se il file e' uno ZIP viene estratto
            if (file.isFile() && nome.endsWith(".zip")) {
                estraiZip(file, dir);
            } else if (file.isDirectory()) {
                //se il file e' una directory lo si analizza
                analisiISO(file.getAbsolutePath(), dir);
            }
        }
    }
}

//metodo che estrae i file ZIP
private void estraiZip(File file, String dir) {
    byte[] buffer = new byte[2048];
    try {
        FileInputStream fis = new FileInputStream(file);
        ZipInputStream zis = new ZipInputStream(fis);
        ZipEntry entry = zis.getNextEntry();

        while(entry != null){
            File nuovo_file = new File(dir + File.separator + entry.getName());
            new File(nuovo_file.getParent()).mkdirs();

            FileOutputStream fos = new FileOutputStream(nuovo_file);
            int len;
            while ((len = zis.read(buffer)) > 0) {
                fos.write(buffer, 0, len);
            }
        }
    }
}

```

```

        fos.close();
        zis.closeEntry();
        entry = zis.getNextEntry();
    }

    zis.closeEntry();
    zis.close();
    fis.close();

} catch (FileNotFoundException e) {
    System.out.println("ERRORE: il file " + file.getName() + " non esiste");
} catch (IOException e) {
    System.out.println("ERRORE di I/O");
}
}

//metodo che recupera i soli file TXT e ne analizza
//lingua, encoding e ID univoco
public void analisiFileTxt(File dir,
    HashMap<Integer, Charset> testi, HashMap<Integer, File> libri) {
    //contiene tutti i file della directory dir
    File[] txt_in_dir = dir.listFiles();

    //calcola il numero di testi TXT analizzati
    int contatore_txt = 0;
    for (File txt : txt_in_dir) {
        //se il file in analisi e' di testo il contatore
        //viene aggiornato e si procede col cercarne
        //lingua, encoding e ID univoco
        if (txt.getName().toLowerCase().endsWith(".txt")) {
            contatore_txt++;

            try {
                BufferedReader reader = new BufferedReader(new
                    FileReader(txt));
                String st = reader.readLine();

                //ferma il ciclo while nel momento in cui ha
                //trovato tutto cio' che occorre per definire il
                //testo. Il controllo e' stato inserito per

```

```

//velocizzare il processo di analisi
boolean fine_controllo = false;

//variabili che definiscono le tre caratteristiche
//del libro che si vogliono trovare
boolean english = false;
int numero_id = 0;
Charset codifica = null;

//finche' non si raggiunge il pattern "****START"
//vengono analizzate le righe una ad una
while (st != null && !fine_controllo) {
    //st viene sempre trasformata tutta in
    //minuscolo per evitare problemi di case
    //sensitivity

    //controlla che la lingua sia inglese
    if (st.toLowerCase().contains("language:")
        && st.toLowerCase().contains("english")) {
        english = true;
    }

    if (st.toLowerCase().contains("[ebook #")) {
        //controlla il numero del testo (ID
        //univoco)
        int indice_hashtag = st.indexOf("#");
        int indice_parentesi= st.indexOf("]");
        numero_id = Integer.parseInt(st.substring(
            indice_hashtag + 1,
            indice_parentesi));
    }

    if (st.toLowerCase().contains("encoding:")){
        //definisce l'encoding giusto per ogni
        //testo
        int indice_due_punti= st.indexOf(":");
        String encoding = st.substring(
            indice_due_punti + 1,
            st.length()).trim();

        try {
            codifica =

```

```

        Charset.forName(encoding);
        //Exception e' generica in modo che
        //includa sottoclassi come
        //IllegalCharsetNamedException
        //e UnsupportedCharsetException
    } catch (Exception e) {
        codifica =
            Charset.defaultCharset();
    }
}

if (st.toLowerCase().contains("***start")) {
    //la riga dopo "***START" segna l'inizio
    //del libro e la fine delle informazioni
    //su di esso
    fine_controllo = true;
}

st = reader.readLine();
}

reader.close();

//se il testo e' in inglese viene aggiunto alla
//HashMap
if (english) {
    //deve essere presente anche la codifica per
    //non incorrere in eccezioni
    if (codifica != null) {
        //se ci fossero due ID uguali verrebbe
        //sovrascritto quello gia' presente.
        //Cosi' si preserva solo una copia
        testi.put(numero_id, codifica);
        libri.put(numero_id, txt);
    }
}

} catch (FileNotFoundException e) {
    System.out.println("ERRORE: il file " + txt.getName() +
        " non esiste");
} catch (IOException e) {

```

```

        System.out.println("ERRORE DI I/O");
    }
}

System.out.println("Numero totale di file txt: " + contatore_txt);
System.out.println("Numero di file txt in inglese da analizzare: "
    + testi.size());
}

//metodo che calcola il numero di parole contenute in tutti i file TXT
//dell'HashMap libri
public void calcolaFrequenze(HashMap<Integer, Charset> testi,
    HashMap<Integer, File> libri, HashMap<String, Integer> frequenze) {

    //numero delle parole
    int parole_totali = 0;
    //contatore per il recupero della codifica di ogni testo
    int contatore = 0;

    for (File txt : libri.values()) {
        //controllo che considera solo il testo in se'
        //togliendo testa e coda
        boolean testa = false;
        boolean coda = false;
        //recupera la codifica per quello specifico testo TXT
        Charset codifica = testi.get(contatore);

        try {
            BufferedReader reader = new BufferedReader(new
                FileReader(txt, codifica));
            String st = reader.readLine();

            while (!testa && st != null) {
                //se trova la testa (***START) comincia il calcolo
                //delle frequenze delle parole
                if (st.toLowerCase().contains("***start")) {
                    testa = true;
                    //finche' non trova la coda (***END) calcola
                    //le frequenze delle parole
                    while (!coda && st != null) {

```

```

        if(st.toLowerCase()
            .contains("***end")) {
            //se trova la coda si ferma
            coda = true;
        } else {
            //Pattern che elimina
            //punteggiatura e caratteri
            //speciali dalle parole rilevate
            Pattern pattern =
                Pattern.compile("[a-zA-Z]+");
            Matcher matcher =
                pattern.matcher(st);

            //analizza le parole ad una ad una
            while (matcher.find()) {
                String parola =
                    matcher.group()
                        .toLowerCase();
                if (!frequenze
                    .containsKey(parola)) {
                    //aggiorna il calcolo
                    //delle parole
                    parole_totali++;
                    //aggiunge la parola
                    //alla HashMap
                    frequenze.put(parola,
                        1);
                } else {
                    //aggiunge una unita'
                    //alla parola gia'
                    //presente
                    frequenze.put(parola,
                        frequenze.get(parola)+
                            1);
                }
            }
        }

        st = reader.readLine();
    }
}

```

```

        } else {
            st = reader.readLine();
        }
    }

    reader.close();

    } catch (Exception e) {
        System.out.println("ERRORE");
    }

    contatore++;
}

System.out.println("Numero totale di parole trovate: " + parole_totali);

}

//metodo che calcola le frequenze delle frequenze (cioe' quante parole sono
//presenti con la stessa frequenza e con quale frequenza)
public void calcolaFrequenzeDelleFrequenze(HashMap<String, Integer> frequenze,
    HashMap<Integer, Integer> frequenze_frequenze) {

    for (int freq : frequenze.values()) {
        if (!frequenze_frequenze.containsKey(freq)) {
            //se la frequenza non e' presente nella HashMap
            //viene aggiunta
            frequenze_frequenze.put(freq, 1);
        } else {
            //se la frequenza e' gia' presente viene aggiornato
            //il numero di parole che occorrono con quella stessa
            //frequenza
            frequenze_frequenze.put(freq, frequenze_frequenze.get(freq)
                + 1);
        }
    }
}

}

}

```


Output

```
import java.util.*;
import java.io.*;

public class Output {
    //metodo che scrive un file CSV con le frequenze
    //di ogni parola trovata
    public void scriviFileFrequenze(String dir, HashMap<String, Integer> frequenze) {
        //crea la cartella che conterra'
        //i file CSV con i risultati
        File directory = new File(dir);
        if (!directory.exists()) {
            directory.mkdirs();
        }

        //crea il file CSV che conterra' i risultati
        File tabella = new File(dir + "\\Frequenza_parole.csv");

        try {
            FileWriter writer = new FileWriter(tabella);
            //HashMap nella quale saranno scritti i valori delle frequenze
            //ordinati
            HashMap<String, Integer> hashmap_ordinata = new
                LinkedHashMap<String, Integer>();
            //ordinamento, in senso decrescente, delle frequenze
            frequenze.entrySet().stream().sorted(Map.Entry
                .comparingByValue(Comparator.reverseOrder()))
                .forEachOrdered(x -> hashmap_ordinata.put(x.getKey(),
                    x.getValue()));

            //scrittura dei titoli delle due colonne del file CSV
            writer.append("Parola").append(";")
                .append("Frequenza").append("\n");

            //scrittura delle frequenze nel file CSV
            for (HashMap.Entry<String, Integer> entry :
                hashmap_ordinata.entrySet()){
                writer.append(entry.getKey()).append(';')
                    .append(entry.getValue().toString()).append("\n");
            }
        }
    }
}
```

```

        writer.flush();
        writer.close();

    } catch (FileNotFoundException e) {
        System.out.println("ERRORE: il file non esiste");
    } catch (IOException e) {
        System.out.println("ERRORE di I/O");
    }
}

//metodo che scrive in un file CSV le frequenze
//delle frequenze delle parole
public void scriviFileFrequenzeFrequenze(String dir, HashMap<Integer,
        Integer> frequenze_frequenze) {
    //crea il file CSV che conterra' i risultati
    File tabella = new File(dir + "\\Frequenza_frequenze.csv");

    try {
        FileWriter writer = new FileWriter(tabella);
        //HashMap nella quale vengono scritti i valori delle frequenze
        //ordinati in ordine decrescente
        HashMap<Integer, Integer> hashmap_ordinata = new
            LinkedHashMap<Integer, Integer>();
        //ordinamento decrescente delle frequenze delle frequenze
        frequenze_frequenze.entrySet().stream().sorted(Map.Entry
            .comparingByKey(Comparator.reverseOrder()))
            .forEachOrdered(x -> hashmap_ordinata.put(x.getKey(),
                x.getValue()));

        //scrittura dei titoli delle due colonne del file CSV
        writer.append("Frequenze delle frequenze")
            .append(";").append("Frequenze").append("\n");

        //scrittura delle frequenze delle frequenze sul file
        for (Map.Entry<Integer, Integer> entry :
            hashmap_ordinata.entrySet()){
            writer.append(entry.getKey().toString()).append(';')
                .append(entry.getValue().toString()).append("\n");
        }
    }
}

```

```
        writer.flush();
        writer.close();

    } catch (FileNotFoundException e) {
        System.out.println("ERRORE: il file non esiste");
    } catch (IOException e) {
        System.out.println("ERRORE di I/O");
    }
}
}
```