Achena Antonio 589563
Di Mauro Gianmarco 587959

Programmazione e Analisi di dati

Relazione progetto: Laboratorio di Programmazione Java

Specifiche inziali e obiettivi

Il progetto ha come obiettivo la scrittura di un programma i linguaggio Java, finalizzato a gestire ed analizzare i dati testuali contenuti in una porzione del database del Progetto Gutenberg; la quantità di dati a disposizione è pari a circa 8,38 Gigabyte. I file si distinguono in diverse estensioni e codifiche, sono compressi in formato .zip e archiviati in cartelle e sottocartelle. La denominazione dei file identifica l'estensione all'interno dell'archivio (es. "`3233_H.zip`" contiene un file html). In particolare, i file testuali presenti negli archivi hanno estensioni: .txt, .rtf, .pdf e .html.

L'obiettivo è estrarre e modificare file di lingua Inglese in formato txt, allo scopo di produrre un'analisi linguistica degli stessi secondo metodi statistici.

Nel programma sono presenti tre *main*, nelle classi *FileWalker, GUTStrip* e *Frequenze* (da eseguire sequenzialmente).

- il *main* presente in *FileWalker* serve a decomprimere i file zip (richiamando *UnzipFiles*);
- il *main* in *GUTStrip* serve a ripulire i file txt;
- il *main* in *Frequenze* serve a contare le frequenze dei token.

In prima istanza viene esplorata la cartella di input; ogni file con estensione .zip, una volta identificato, viene decompresso nella cartella *Unzipped* mediante la classe *FileWalker,* che richiama la classe *UnzipFiles.* Una volta decompressi, i file vengono ripuliti da parti superflue ai fini dell'analisi tramite la classe *GUTStrip*, la quale elimina, secondo consegna del progetto, le parti prima di "`*** START OF`" e dopo "`*** END OF`", creando una nuova cartella con i documenti decompressi ripuliti, chiamata *Output*.

Successivamente, la classe *Frequenze,* analizzando singolarmente tutti i file, conterà le frequenze delle parole e le salverà in una HashMap, scrivendo progressivamente in un file .txt i valori.

Scelte di implementazione

I file che il programma processerà hanno formato .txt, scritti in lingua Inglese, e sono situati nelle cartelle dalla 0 alla 9; le fasi di lavoro sono tre: *decompressione*, *pulizia* del testo e infine *conteggio* della frequenze per la creazione di un vocabolario. Queste fasi sono separate, con tre rispettivi *main* per ottimizzare l'uso delle classi, l'uso della memoria e il tempo d'esecuzione totale del programma.

Al fine di visualizzare in maniera ottimale il conteggio delle parole, si è optato per la scrittura su file di testo dei risultati, onde evitare eventuali problemi di visualizzazione nella stampa a schermo.

Per la struttura dati si è optato per ll'utilizzo di una *HashMap*, che permette una gestione migliore e più rapida per la ricerca di elementi.

Achena Antonio 589563
Di Mauro Gianmarco 587959

Esempi di funzionamento

Il ciclo di esecuzione del programma inizia dalla classe *FileWalker;* l'accesso ai file .zip contenuti nelle sottocartelle avviene con procedimento ricorsivo: nel caso in cui venga esaminata una cartella, viene richiamata la funzione *walk*, altrimenti viene chiamata la funzione *unZipIt*. Questo consente di esplorare il corpus, in modo da poter selezionare i file compressi in formato.zip nelle cartelle, processarli e rendere possibile la successiva operazione di modifica del contenuto testuale.

In due fasi, ci si avvale di due cartelle temporanee per raccogliere con ordine, rispettivamente, prima i file decompressi nella propria cartella di denominazione, e in seguito la loro versione finale di output, con testo ripulito.

La classe *UnzipFiles* è dedicata ad eseguire la decompressione dei soli file con estensione .txt, tramite un controllo del formato nella denominazione del file selezionato.

Figura 1: Comparazione tra stato iniziale e finale del file 10001.ZIP, prima e dopo la decompressione

Per eliminare le parti di testo non utili all'analisi nella classe *GUTStrip* viene eseguito un controllo che permette di considerare il testo dopo "*** START OF" e prima di "*** END OF", scrivendo la porzione di file corretta in nuovo documento ripulito privo degli elementi di catalogazione ridondanti e superflui, permettendo così la creazione di un corpus più adatto all'analisi statistica.

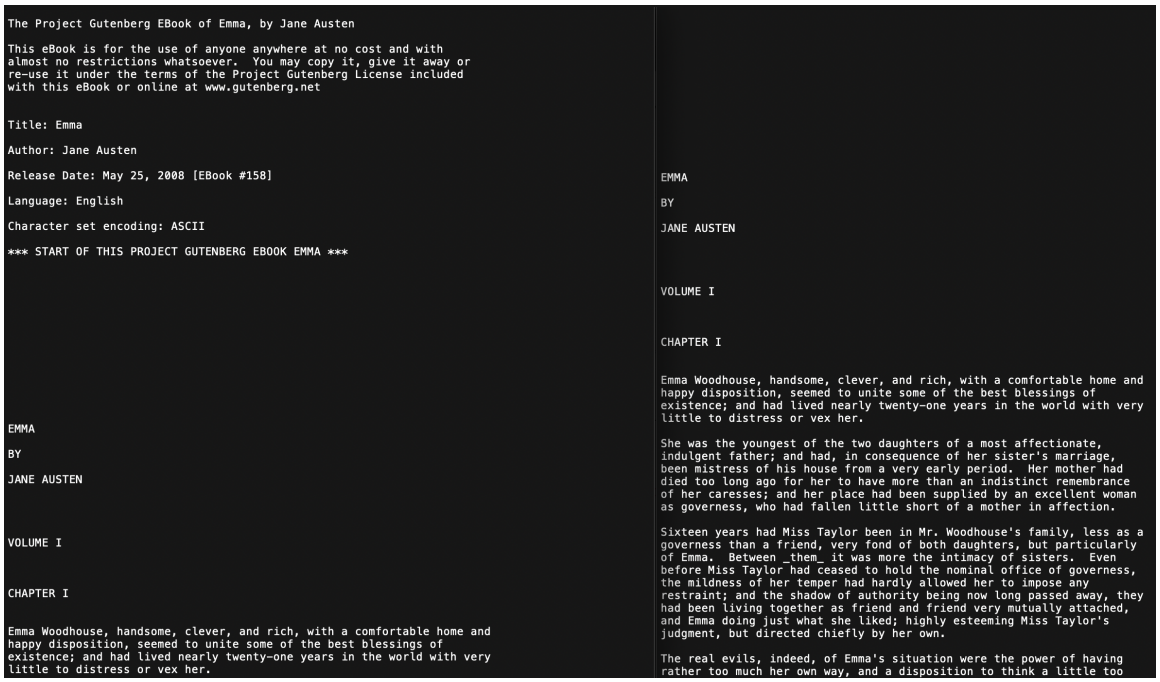

Figura 2: Comparazione tra stato iniziale e finale del file 158.txt, scelto casualmente per la verifica

Achena Antonio 589563
Di Mauro Gianmarco 587959

Nella classe *Frequenze*, vengono conteggiate le parole di un testo inserendo, se non già inserito in precedenza, un nuovo termine nella *HashMap*. Verrà poi preso in considerazione un nuovo file, ripetendo questo processo, fino alla creazione di una *HashMap* completa che riporta una lista di coppie <chiave, valore>, ovvero <*token*, *frequenza*>

Nella seguente tabella vengono riportate secondo le rispettive frequenze i primi 75 *token,* estratti dal file di output, in ordine decrescente. Sono presenti alcune ripetizioni, dovute alla distinzione delle lettere maiuscole e minuscole per parole identiche.

the	74129972
of	43623590
and	38446416
to	32972569
a	24815981
in	21121383
I	14408385
that	14268363
was	13581283
he	10938045
his	10641079
it	10494243
is	9673187
with	9568113
as	9029961
for	8939602
had	8037435
be	7172583
not	7011907
you	6981250
on	6756276
at	6549431
her	6523719
The	6485616
by	6402536

s	6370282
which	5860069
have	5489578
from	5226564
but	5023009
him	4852366
this	4614014
were	4568075
all	4538493
they	4446515
she	4392598
or	4309293
are	4050428
my	3815471
one	3774626
their	3748160
an	3629850
so	3540482
me	3534034
we	3359573
said	3270132
them	3209393
been	3182402
would	3120369
who	3114308

He	3063020
no	2768569
will	2691173
there	2670817
It	2613304
when	2576801
out	2496701
t	2452589
up	2430215
more	2401632
if	2368678
into	2258617
has	2139367
And	2085273
could	2080119
But	2061540
what	2052988
man	2036127
some	1981817
do	1953838
very	1935633
time	1916974
than	1908307
its	1844965
our	1808104

Dopo la creazione del corpus completamente decompresso e ripulito, la classe *Frequenze* procede alla creazione di tre file testuali di output, al fine di eseguire il conteggio di *types* e di *token*, l'indicizzazione e il ranking delle frequenze nel corpus, così da poter utilizzare questi dati salvati per condurre analisi statistiche rilevanti sui testi che fanno parte del campione.

Achena Antonio 589563
Di Mauro Gianmarco 587959

Analisi e considerazioni sui risultati

Nella fase di conteggio delle frequenze, l'ammontare dei file raggiungeva un peso di circa 7,33 GB, per un totale di 26.094 file.

Il costruttore nella classe *Frequenze*, avvalendosi dei metodi ausiliari *add*, *read* e della classe *ScanDirectory*, calcola il dizionario di frequenza e permette di ottenere i dati relativi al vocabolario e al corpus, utili per studiare i testi in relazione alle leggi di Zipf e Heaps.

Ottenuto il dizionario di frequenza, si ha l'output del programma, costituito delle coppie del tipo <rango, frequenza>, da rappresentare graficamente in scala doppio logaritmica. Il rango aumenta in ordine crescente ed è proporzionale alla frequenza dei *token*: la parola con *rango = 1* è quella con frequenza massima. La rappresentazione grafica mostra come il corpus segua effettivamente l'andamento della legge di Zipf: si osservano un numero ristretto di parole che ricorrono tantissime volte, mentre la maggior parte dei token ha una frequenza bassa o medio-bassa. La coda della curva indica gli *hapax*, ovvero parole con *frequenza = 1,* che rappresentano quell'insieme di parole che caratterizzano maggiormente il dominio semantico e il lessico di un documento o un corpus.

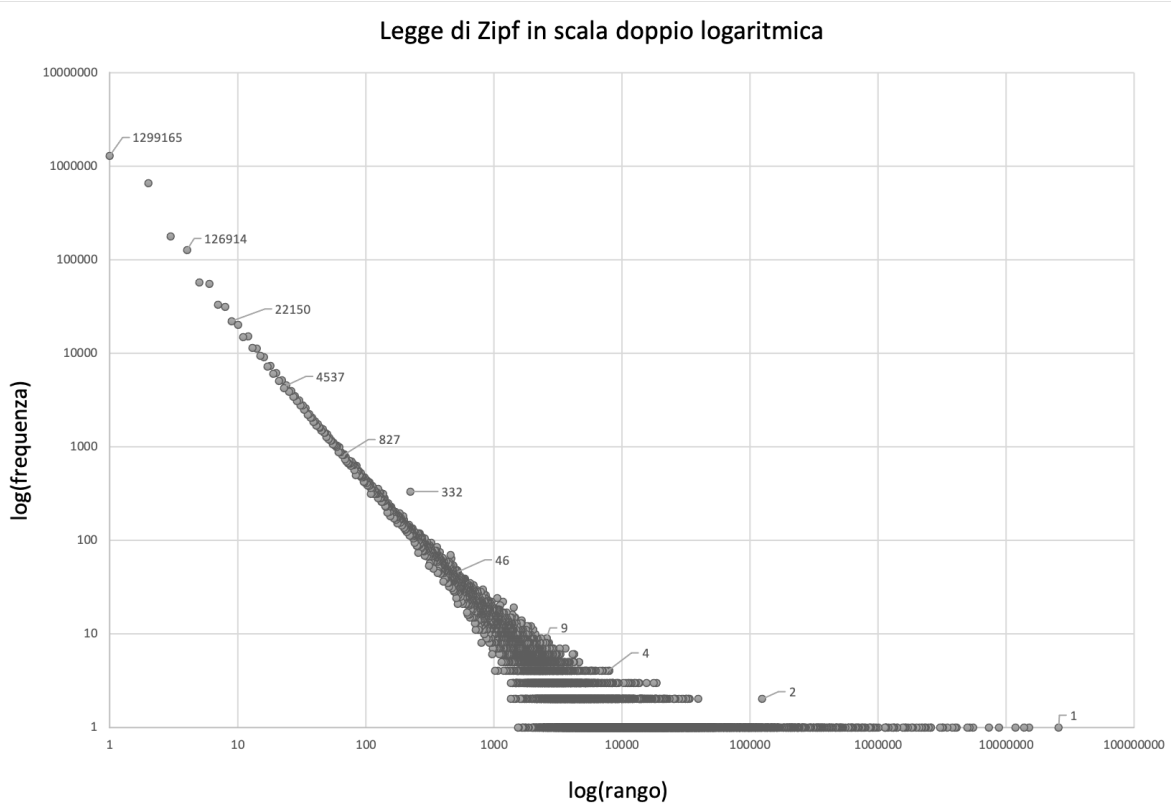

Figura 3: Grafico con etichette "frequenze di frequenze", Legge di Zipf in scala doppio logaritmica

Achena Antonio 589563
Di Mauro Gianmarco 587959

La legge di Heaps afferma che al crescere della quantità di termini di testo identificati sarà proporzionale un livello decrescente di scoperta del vocabolario completo da cui sono tratti i termini distinti.

Dopo aver eseguito un ciclo completo del programma, viene prodotto in output un file .txt che riporta delle coppie del tipo <grandezza del vocabolario, grandezza del corpus> per ogni token. Il rapporto tra queste due grandezze (*type* e *token*) è indice di ricchezza lessicale e, nel grafico in Figura 4, è possibile seguire l'andamento dell'indice al crescere del corpus.

Il vocabolario cresce in maniera costante seguendo con buona approssimazione la linea di tendenza di Heaps; vi sono alcuni punti in cui la curva non è continua, specificamente nella zona centrale, dovuti probabilmente a particolari varietà linguistiche e testuali presenti nei documenti analizzati.

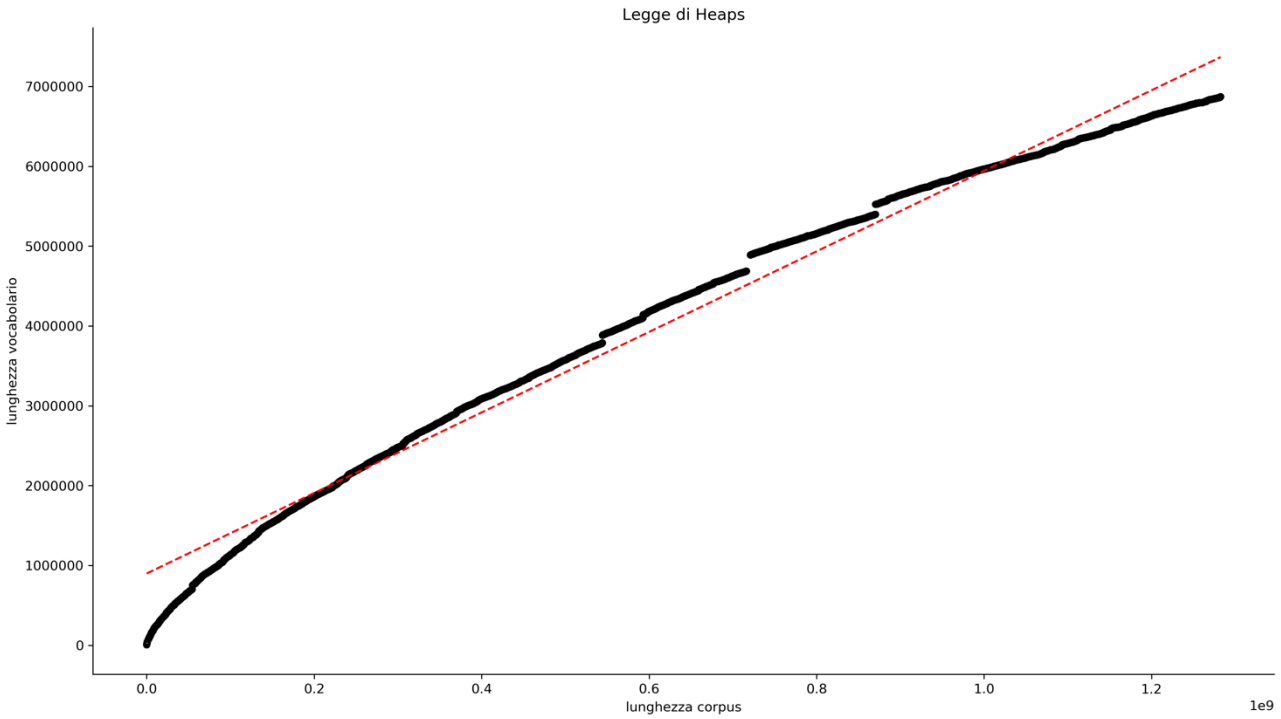

Figura 4: Grafico con trendline, Legge di Heaps

Achena Antonio 589563
Di Mauro Gianmarco 587959

Listato dei programmi

`FileWalker`

La classe si occupa di decomprimere i file .zip, richiamando la classe *UnzipFiles*. Per accedere ai file .zip contenuti nelle sottocartelle, viene usato il meccanismo della ricorsività: se viene presa in esame una cartella, viene richiamata la funzione *walk*, altrimenti, trattandosi di un file, viene chiamata la funzione *unZipIt*. Inoltre, nel *main* è compresa la stampa del tempo impiegato espresso in secondi.

È presente un metodo ausiliario booleano per la selezione delle cartelle dalla 0 alla 9, ignorando quelle denominate alfabeticamente.

```
import java.io.File;
import java.io.IOException;

public class FileWalker {
	// indirizzi delle cartelle, modificare i separatori in base al sistema operativo
	static String inputDir = "/Users/User/pgdvd042010";
	static String unzipDir = "/Users/User/Gutenberg Project/Unzipped";
	static String outputDir = "/Users/User/Gutenberg Project/Output";

	public void walk(String path) {

		File root = new File(path);
		File[] list = root.listFiles();
		String appoggio;
		UnzipFiles unZip = new UnzipFiles();
		if (list == null)
			return;

		for (File f : list) {
			if (f.isDirectory()) {
				String name = f.getName();
				if (isNumeric(name)) {
					walk(f.getAbsolutePath());
				}// richiama la funzione walk o unzipit in base alla denominazione del file
			} else {
				appoggio = f.getAbsoluteFile().toString();
				if (appoggio.substring(appoggio.length() - 4).equals(".ZIP")) {
					unZip.unZipIt(appoggio, unzipDir);
				}
			}
		}
	}

	public static void main(String[] args) throws IOException {
		FileWalker fw = new FileWalker();
		long inizio = System.currentTimeMillis();
		// l'indirizzo della cartella da cui prendere i file in input
		fw.walk(inputDir);
		System.out.println("Unzip terminato!");
		long fine = System.currentTimeMillis();
		long tempoVero = (fine - inizio) / 1000;
		System.out.println(" Impiegati " + tempoVero + " secondi ");
	}

	public static boolean isNumeric(final String str) {
		// null or empty
		if (str == null || str.length() == 0) {
			return false;
		}

		for (char c : str.toCharArray()) {
			if (!Character.isDigit(c)) {
				return false;
			}
		}
		return true;

	}
}
```

Achena Antonio 589563
Di Mauro Gianmarco 587959

UnzipFiles

Questa classe si occupa di decomprimere i file .zip, creando una cartella in cui inserirli. Verranno presi soltanto i file .txt mediante un controllo di stringa.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

public class UnzipFiles {
    List<String> fileList;

    public void unZipIt(String zipFile, String outputFolder){

    byte[] buffer = new byte[1024];

    try{
       // crea una cartella output nel caso non esista
       File folder = new File(outputFolder);
       if(!folder.exists()){
              folder.mkdir();
       }

       // estrae contenuto del file zip
       ZipInputStream zis = new ZipInputStream(new FileInputStream(zipFile));

       // ottiene la lista dei file da zippare
       ZipEntry zip = zis.getNextEntry();

       while(zip!=null){

          String fileName = zip.getName();
          File newFile = null;
          if(fileName.substring(fileName.length()-4).equals(".txt")) {
// controlla il tipo txt nella denominazione dei file

                        newFile = new File(outputFolder + File.separator + fileName);
             // crea cartella quando ci sono altre cartelle all'interno del file,
              altrimenti restituisce "errore FileNotFoundException"
                        new File(newFile.getParent()).mkdirs();

                        // creazione del file decompresso
                        FileOutputStream fos = new FileOutputStream(newFile);
                        int len;
                        while ((len = zis.read(buffer)) > 0) {
                               fos.write(buffer, 0, len);
                        }

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

          }
          catch(IOException ex){
             ex.printStackTrace();
          }

    }
}
```

Achena Antonio 589563
Di Mauro Gianmarco 587959

ScanDirectory

Questa classe si occupa di fornire la lista dei file che saranno utilizzati nelle classi *Frequenze* e *GUTStrip*.

```
import java.io.File;
import java.io.FilenameFilter;

public class ScanDirectory implements FilenameFilter {
	private String ends;
	private String[] list;

	// nome directory, suffisso usato per filtrare file
	public ScanDirectory(String dirName, String ends) {
		this.ends = ends;
		// array di file filtrati secondo FilenameFilter
		this.list = new File(dirName).list(this);
	}

	public ScanDirectory(String dirName) {
		this(dirName, null);
	}

	public String[] getList() {
		return list;
	}

	@Override
	public boolean accept(File dir, String name) {
		if (ends == null)
			return true;
		return name.toLowerCase().endsWith(ends.toLowerCase());
	}

}
```

Achena Antonio 589563
Di Mauro Gianmarco 587959

GUTStrip

Questa classe si occupa di ripulire i file dalle parti di testo non utili all'analisi. Anche in questo caso, è stata inserita una stampa del tempo impiegato, espresso in secondi.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class GUTStrip {
        private String[] fileList;
        // indirizzi delle cartelle, modificare i separatori in base al sistema operativo
        static String inputDir = "/Users/User/pgdvd042010";
        static String unzipDir = "/Users/User/Gutenberg Project/Unzipped";
        static String outputDir = "/Users/User/Gutenberg Project/Output";


        public GUTStrip() {
                fileList = new ScanDirectory(unzipDir , ".txt").getList();
                if (fileList == null)
                        throw new RuntimeException("Nessun file trovato");
        }

        public void strip() {
                try {
                        new File(outputDir).mkdir();
                        String[] x = new ScanDirectory(outputDir, ".txt").getList();
                        if (x != null && x.length != 0) {
                                System.out.println("Esistono già dei file ripuliti");
                                return;
                        }
                        for (int c = 0; c < fileList.length; c++) {
                                BufferedReader br;
                                PrintWriter out;
                                String line = "";
                                // modificare i separatori in base al sistema operativo
                                String filePath = unzipDir + "/" + fileList[c];
                                String newFilePath = outputDir +"/"+ fileList[c];

                                // creazione della directory
                                br = new BufferedReader(new FileReader(filePath));

                                out = new PrintWriter(new BufferedWriter(new FileWriter(newFilePath, true)));
                                boolean skip = true;
                                while ((line = br.readLine()) != null) {

                                        if (!skip)
                                                out.println(line);
                                        if (line.startsWith("***START OF") || line.startsWith("*** START OF"))
                                                skip = false;
                                        if (line.startsWith("***END OF") || line.startsWith("*** END OF"))
                                                break;
                                }
                                br.close();
                                out.close();
                        }
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                System.out.println("Pulizia dei file terminata!");
        }
        public static void main (String [] args) {
                GUTStrip pulizia = new GUTStrip();
                pulizia.strip();
        }
}
```

Achena Antonio 589563
Di Mauro Gianmarco 587959

Frequenze

La classe *Frequenze* è dedicata al conteggio dei *type* e dei *token* presenti nei diversi file, inserendoli nella *HashMap* e stampandoli su file. La classe utilizza *ScanDirectory*.

```
import java.io.*;
import java.math.BigDecimal;
import java.util.*;
import java.util.stream.Collectors;
public class Frequenze {// modificare i separatori in base al sistema operativo
          static String inputDir = "/Users/User/Downloads/pgdvd042010";
          static String outputDir = "/Users/User/Gutenberg Project/out";
          final static String delim =" ,.;?!\"\'";      // delimitatori delle parole
private Vector<Integer> vocabularySize = new Vector<> ();
          private Vector<Integer> corpusSize = new Vector<> ();
          public int nToken=0;
          public Frequenze() throws IOException{
                    // modificare i separatori in base al sistema operativo
                    String indirizzoFile=inputDir+"/"+ "Freq.txt"; // crea file Token-Freq.
                    String indizizzoZipf=inputDir+"/"+ "Zipf.txt"; // crea file Rank-Freq.
                    String indirizzoHeaps=inputDir+"/"+ "Heaps.txt"; // crea file Freq.-Freq.
                    PrintWriter outFile = new PrintWriter(new BufferedWriter(new
                                        FileWriter(indirizzoFile, true)));
                    Map<Object,Integer> h = new HashMap<Object,Integer>();
                    ScanDirectory sd = new ScanDirectory(outputDir,".txt");
                    for(String fileName : sd.getList()) read(fileName, h);
                    Map<Object, Integer> frequencyDictionarySorted = h.entrySet().stream().sorted(Collections.reverseOrder(Map.Entry
                                        .comparingByValue())).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (e1,e2) -> e2,
LinkedHashMap::new));
                    try {
                              for (Map.Entry e : frequencyDictionarySorted.entrySet()) {
                                        outFile.write(e.getKey() + "\t" + e.getValue() + "\n");
                              }
                    } catch(Exception e) {
                                        e.printStackTrace();
                    }
                    outFile.close();
                    try {
                              PrintWriter Zipf = new PrintWriter(new BufferedWriter(new
                                                  FileWriter(indizizzoZipf, true)));
                              final Vector<Integer> valori = new Vector<> ();

                              for (Map.Entry e : frequencyDictionarySorted.entrySet()) {
                                        valori.add((int)e.getValue());
                              }
                              int frequenza = 1;
                              int i = 0;
                              while (i < valori.size()) {
                                        if (i != valori.size()-1) {
                                                  if ((int)valori.get(i) == (int)valori.get(i+1)) {
                                                            frequenza++;
                                                            i++;
                                                  } else {
                                                            Zipf.write(frequenza + "\t" + valori.get(i) + "\n");
                                                            frequenza = 1;
                                                            i++;
                                                  }
                                        } else {
                                                  Zipf.write(frequenza + "\t" + valori.get(i) + "\n");
                                                  frequenza = 1;
                                                  i++;
                                        }
                              }
                              Zipf.close();
                    } catch(Exception e) {
                              e.printStackTrace();
                    }

                    try {
                              PrintWriter Heap = new PrintWriter(new BufferedWriter(new
                                                  FileWriter(indirizzoHeaps, true)));
                              for (int i = 0; i < vocabularySize.size();i++) {
                                        Heap.write(vocabularySize.get(i) + "\t" + corpusSize.get(i) + "\n");
                              }
                              Heap.close();
                    } catch(Exception e) {
                              e.printStackTrace();
                    }
          }
          void add(Map<Object,Integer> m, Object v){
                    Integer o = m.get(v);
                    if(o==null) m.put(v,1);
                    else m.put(v,o+1);
          }
          void read(String inp, Map<Object,Integer> h){
          try {
                    BufferedReader in = new BufferedReader(new FileReader(outputDir+"/"+inp));
                    String line = in.readLine();
                    while(line!=null){
                              StringTokenizer st = new StringTokenizer(line, delim);
                              while(st.hasMoreTokens()){
                                        add(h, st.nextToken());
                                        nToken++;
                              }
                              line = in.readLine();
                    }
                    vocabularySize.add(h.size());
                    corpusSize.add(nToken);
          } catch(IOException e) {
                    e.printStackTrace();
          }
          }
          public static void main (String [] args) throws IOException {
           Frequenze f=new Frequenze();
          }
}
```