Francesco Kotopulos De Angelis – Matricola: 549045 – Pisa, 26/11/2017

Primo progetto intermedio - Programmazione 2

Compilazione ed esecuzione

Il progetto è costituito da due Main, uno per ciascuna delle due diverse implementazioni richieste.

`MainHT` lavora su una HashTable, mentre `MainTM` utilizza come struttura dati una TreeMap.

Compilato il codice da terminale con il comando `JAVAC`, all'esecuzione verrà richiesto di inserire il percorso del file di testo (.txt) da analizzare.

Struttura del codice

La prima parte del codice è interamente dedicata alla lettura di un file di testo. È stato creato un FileReader e un BufferedReader per poter leggere l'input riga per riga. Fintanto che esiste una sequenza di parole da leggere dal file di ingresso, si chiama il metodo `replaceAll` sulla stringa `"input"` appena letta per rimuovere tutto ciò che non faccia match con caratteri dell'alfabeto minuscolo e maiuscolo, quindi tutta la punteggiatura `([^a-zA-Z])`. Successivamente si trasforma la stringa in caratteri minuscoli e si chiama il metodo split per inserire tutte le parole separate da uno spazio in un array di stringhe.

Creato l'array `(arrInput)`, si richiama il metodo `incCount` per inserire ogni stringa dell'array nella struttura dati. Il metodo `incCount` prende una chiave di tipo `<E>` come parametro in ingresso e se la chiave è stata precedentemente inserita, viene incrementato solamente il contatore delle occorrenze legato alla chiave in questione. Se quest'ultima non appartiene alla struttura dati, allora viene inserita per la prima volta con contatore delle occorrenze pari a 1.

Nel ciclo `while` usato per leggere il testo in ingresso, si utilizza anche una stringa `"testo"` che concatena tutte le righe in input lette. Inoltre, è presente una variabile intera `"words"` che conta il numero totale di parole inserite.

Dopo aver riempito la mia struttura dati, si richiama il metodo `getIterator`, che restituisce un iteratore sulle chiavi inserite, ordinato in base alle occorrenze delle chiavi. In caso di chiavi con la stessa occorrenza, prevale l'ordine naturale delle chiavi, nel nostro caso, l'ordine alfabetico.

Nell'ultima parte del codice viene stampato il numero di caratteri con e senza spazi, utilizzando la stringa `"testo"` precedentemente memorizzata. Per avere il numero di caratteri senza spazi è sufficiente chiamare una `replaceAll` sulla stringa `"testo"` imponendo un match con tutti gli spazi e sostituendoli con stringa vuota. Infine, il numero totale di parole è memorizzato dalla variabile intera `"words"`.

Nell'ultimo passaggio scorro l'iteratore ordinato e fintanto che si avrà un altro elemento nell'iteratore `(i.hasnext != null)` verrà stampato quell'elemento con le sue occorrenze recuperate dalla struttura dati.

Francesco Kotopulos De Angelis – Matricola: 549045 – Pisa, 26/11/2017

Scelte implementative

Nel progetto sono presenti due classi corrispondenti a due diverse implementazioni dell'interfaccia `DataCounter <E>`**.**
`MyDataCounterHT` è l'implementazione dell'interfaccia che utilizza una HashTable come struttura dati.
`MyDataCounterTM` è l'implementazione dell'interfaccia che utilizza una TreeMap come struttura dati.

In entrambe le implementazioni sono presenti tutti i metodi richiesti dall'interfaccia.

incCount: riceve come parametro in ingresso una chiave di tipo `<E>`. Se questa appartiene alla struttura dati, viene incrementato il contatore delle occorrenze relative alla chiave. Diversamente, verrà inserita nella struttura dati una nuova chiave con occorrenza pari a 1.

getCount: riceve come parametro in ingresso una chiave di tipo `<E>`. Se la chiave appartiene alla struttura dati, viene restituito il valore delle occorrenze relative alla chiave in questione, se no, verrà restituito il valore intero 0.

getSize: non ha nessun parametro in ingresso e restituisce la dimensione della struttura dati, ovvero il numero delle chiavi presenti in essa.

getIterator: restituisce un iteratore, che non supporta il metodo `remove`, sulle chiavi della struttura dati, ordinato in base alle loro occorrenze. Per creare questo iteratore si è definita una nuova classe `MyIterator.`

Classe MyIterator

In questa classe sono stati definiti due costruttori che prendono in ingresso una HashTable o una TreeMap. Tuttavia, in entrambi i casi, viene creato un keyset delle strutture dati e copiato in una struttura ordinabile come può esserlo una ArrayList. Si chiama il metodo `sort` sulla lista con un comparatore definito che confronti le occorrenze delle chiavi. Le chiavi con le stesse occorrenze vengono ordinate in base all'ordine naturale delle chiavi di tipo `<E>`. Successivamente si richiama il metodo `iterator` per restituire un iteratore sulle chiavi della lista.

Il metodo remove, se richiamato, lancia una UnsopportedOperationException (eccezione di tipo Unchecked, presente in Java).