

Corso di Algoritmi e Strutture Dati—Informatica per il Management

Progetto finale, versione 1.1, 8/9/2010

(Questa specifica può essere soggetta a revisioni: ogni modifica verrà segnalata nelle pagine del corso)

Revisioni:

Rev. 1.1, 8/9/2010: Ho specificato che il carattere “\n” indica l'a capo (*carriage return*), NON deve quindi essere interpretato letteralmente come la coppia di caratteri “\” e “n”.

Specifiche del progetto

Lo scopo del progetto è quello di realizzare un mini-motore di ricerca in Java in grado di operare su una collezione di file di testo ospitati sul proprio PC. In breve, il programma deve indicizzare opportunamente (i dettagli verranno dati a breve) uno o più files di testo, e deve essere in grado di fornire all'utente l'elenco dei files che contengono una o più parole date in input. Sia l'operazione di indicizzazione, sia la fase di ricerca dei files contenenti le parole date devono essere realizzate in maniera efficiente. Il motore di ricerca deve essere una applicazione di tipo puramente testuale: deve accettare input e produrre output esclusivamente su console; l'applicazione non deve fornire nessun tipo di interfaccia grafica.

In dettaglio, l'applicazione deve accettare sulla riga di comando una lista di n files, che chiameremo F_1, F_2, \dots, F_n , con $n \geq 1$. Ciascun file contiene una lista di parole, in cui ciascuna parola può comparire più volte nello stesso file, e può comparire in files diversi: scopo dell'applicazione è indicizzare gli n files, consentendo all'utente di eseguire delle query sulla collezione di documenti. Una query Q è rappresentata da una lista di k parole $Q = \{w_1, w_2, \dots, w_k\}$ con $k \geq 1$. Non è detto che le parole in Q appartengano necessariamente a qualche file. Diciamo che un file F soddisfa la query Q se e solo se F contiene tutte le parole presenti in Q . A fronte di una query Q , il programma deve fornire come output la lista dei files che soddisfano Q , ordinando il risultato in ordine decrescente in base al numero complessivo di match. Formalmente, se un file F contiene la parola w , sia $\text{count}(F, w)$ il numero di occorrenze di w in F . Definiamo $\text{count}(F, Q)$ come la somma delle occorrenze di tutte le parole $w \in Q$ nel file F , ossia $\text{count}(F, Q) = \sum \text{count}(F, w_i)$. La lista dei files F che soddisfano Q deve essere ordinata in ordine decrescente in base al valore di $\text{count}(F, Q)$. In caso di parità, i files devono essere elencati in ordine lessicografico sul nome.

Questa è la specifica dettagliata del programma

1. Il programma deve accettare comandi dallo standard input, e deve produrre risultati sullo standard output.
2. Il programma (che supponiamo si chiami `indexer`) accetta una lista di files sulla riga di comando. Ad esempio:

```
java -jar indexer.jar file1.txt file2.txt file3.txt file4.txt
```

In generale possono venir forniti un numero arbitrario di files sulla riga di comando. Viene sempre fornito almeno un file. Se uno dei files non esiste, il programma deve stampare un messaggio di errore e terminare.

3. Ciascun file contiene una lista di parole; ogni riga contiene una singola parola, che si assume composta da una sequenza di sole lettere minuscole (non sono ammesse lettere maiuscole, spazi né altri simboli; non è richiesto di effettuare il controllo che effettivamente le parole contengano solo lettere minuscole). Ogni parola è seguita dal carattere di “a capo” (`\n`). **Nota: il carattere `\n` rappresenta l' “a capo” (return), quindi NON va interpretato come la coppia di simboli**

"\" e \"n\". Si può assumere che tutti i files contengano almeno una parola (non è richiesto di effettuare il controllo che i files siano non vuoti).

4. A questo punto il programma resta in attesa di comandi dallo standard input.
5. Per terminare il programma è necessario digitare la cifra zero, seguita da return:
0\n
6. Se viene inserito un numero intero positivo k (seguito da \n), il programma resta in attesa di k ulteriori righe di input dallo standard input, ciascuna contenente una parola terminata da \n. Ad esempio:
4\n
pippo\n
pluto\n
paperino\n
topolino\n

Questo richiede al programma la lista dei nomi dei files, tra quelli forniti sulla riga di comando, che contengono tutte le parole indicate. Se nessun file soddisfa la query, il programma non stampa nulla.

7. Per ogni file F che soddisfa la query Q, il programma deve stampare una riga che contiene il nome del file F, seguito dal carattere spazio, seguito dal valore di count(F,Q). I files che soddisfano le query devono comparire in ordine decrescente rispetto al corrispondente valore di count(F,Q). In caso di parità, i files devono essere elencati in ordine lessicografico crescente sul nome (ossia un ipotetico file 'aaa.txt' viene elencato prima di 'aab.txt').

Esempio

Consideriamo tre files che chiamiamo primo.txt, secondo.txt e terzo.txt. Il loro contenuto è come segue:

primo.txt:

```
pippo\n  
pluto\n  
paperino\n  
topolino\n
```

secondo.txt:

```
pippo\n  
pluto\n  
pippo\n
```

terzo.txt:

```
topolino\n  
minnie\n  
topolino\n  
paperino\n
```

```
minnie\n
paperino\n
```

Supponendo che il programma venga invocato con `java -jar indexer.jar primo.txt secondo.txt terzo.txt`, mostriamo di seguito una ipotetica sessione di lavoro (le righe in grassetto rappresentano l'output del programma):

```
1\n
```

```
minnie\n
```

```
terzo.txt 2 (compaiono due occorrenze di "minnie" nel file terzo.txt)
```

```
2\n
```

```
pippo\n
```

```
paperino\n
```

```
primo.txt 2 (compare una occorrenza di "pippo" e una occorrenza di "paperino"--totale 2 occorrenze complessive—nel file primo.txt)
```

```
2\n
```

```
topolino\n
```

```
paperino\n
```

```
terzo.txt 4 (2 occorrenze di "topolino" e 2 occorrenze di "paperino" nel file terzo.txt)
```

```
primo.txt 2
```

```
1\n
```

```
topolino\n
```

```
terzo.txt 2
```

```
primo.txt 1
```

```
1\n
```

```
gambadilegno\n
```

```
(il programma non stampa nulla)
```

```
0\n
```

```
(il programma termina)
```

Cosa Consegnare

È necessario consegnare i sorgenti del programma e una breve relazione che lo descrive. La relazione non dovrebbe superare le 10 facciate; chi ha necessità di scrivere una relazione più lunga, è pregato di contattare il docente. I sorgenti devono essere commentati in maniera adeguata (senza esagerare!). Deve essere fornita la spiegazione di come compilare i sorgenti; idealmente, dovrebbe essere fornito un opportuno Makefile (o build.xml per Ant) per la compilazione semiautomatica. La relazione, in formato PDF, deve contenere cognome, nome e numero di matricola dell'autore o degli autori. Deve altresì contenere una spiegazione esauriente del funzionamento del programma, con particolare riguardo alla descrizione degli algoritmi e delle strutture dati impiegate. Deve indicare la complessità computazionale di tutte le fasi di funzionamento dell'algoritmo. Ove possibile, si indichi la complessità

nel caso pessimo e nel caso ottimo.

Il progetto va consegnato inviando una mail a marzolla@cs.unibo.it con titolo: “Consegna progetto ASD 2010”. Nella mail indicare il nome, cognome e numero di matricola dell'autore (o degli autori), includere la relazione in formato PDF, e i sorgenti in un archivio .zip oppure .tar.gz. I sorgenti NON devono contenere il programma precompilato: il .jar verrà generato dal docente seguendo le istruzioni fornite.

Come verrà valutato il progetto

Il progetto verrà valutato in base ai seguenti criteri:

- Rispetto della specifica data in questo documento;
- Correttezza del programma consegnato rispetto alla specifica;
- Chiarezza del codice sorgente;
- Efficienza degli algoritmi e delle strutture dati adottate;
- Qualità e completezza della relazione che accompagna il progetto;
- Correttezza delle risposte orali fornite ai chiarimenti eventualmente richiesti dal docente.

Regole generali

I programmi devono essere scritti in Java 1.5; verranno testati sulle macchine del laboratorio (in ambiente Linux).

Il progetto può essere svolto singolarmente o in coppia. Naturalmente, i progetti svolti in coppia verranno valutati con criteri più stringenti rispetto a quelli svolti singolarmente. Ciascun progetto deve essere svolto indipendentemente dagli altri. Progetti copiati in tutto o in parte verranno annullati, e gli autori dovranno consegnare un nuovo programma, basato su nuove specifiche.

È consentito l'uso di strutture dati o algoritmi forniti dalla libreria standard Java. È anche consentito l'uso di codice (open source) presente in rete, purché l'origine del codice sia chiaramente indicata nella relazione. I progetti che fanno uso di codice presente in rete senza citarne l'origine saranno annullati, e gli autori dovranno consegnare un nuovo programma basato su nuove specifiche.

Agli autori dei progetti possono essere richieste spiegazioni dal docente. Tutti autori dei progetti devono essere in grado di rispondere a qualsiasi tipo di domanda su tutto il codice consegnato, sia che sia stato scritto dagli autori sia che si tratti di software libero recuperato in rete.