Progetto di Algoritmi e Strutture Dati - A.A. 2013-14

Versione 0.7 del 15/01/14

Esercizio 1

L'associazione "W la ricerca" decide di memorizzare per ognuno dei propri iscritti i seguenti dati:

- nome e cognome
- data e luogo di nascita.

Ad ogni socio dovrà poi essere attribuito un codice identificativo univoco.

La memorizzazione dovrà avvenire nel seguente modo: i soci il cui cognome inizia con la stessa lettera verranno inseriti in una lista (che dovrà essere mantenuta ordinata rispetto al cognome, e in caso di cognomi uguali, rispetto al codice identificativo). Le liste (una per ogni lettera) dovranno poi essere inserite in un array (ogni posizione dell'array conterrà la reference ad una lista).

Implementare i metodi per l'inserimento, la ricerca (per cognome), la cancellazione e la stampa (in ordine alfabetico rispetto al cognome) di tutti i soci dell'associazione.

Stimare complessità delle operazioni di inserimento, ricerca e cancellazione nel caso ottimo e nel caso pessimo.

Note implementative

Il file che contiene il main si dovrà chiamare Esercizio1.java.
I dati da inserire dovranno essere letti dal file InputEs1.txt che avrà il seguente formato:

Mario
Rossi
Bologna
05/12/1990

Ogni socio sarà separato dal successivo da una linea vuota.

I dati per la ricerca dovranno essere letti dal file RicercaEs1.txt che conterrà, separati da una linea vuota, i cognomi dei soci da ricercare. I risultati della ricerca dovranno essere stampati nel file OutputRicercaEs1.txt con formato:

Mario
Rossi
Bologna
05/12/1990
<codice identificativo>

Nel caso di due o più soci con lo stesso cognome il file dovrà riportare i dati di entrambi i soci, separati da una linea vuota.

I dati per la cancellazione dovranno essere letti dal file CancellazioneEs1.txt che conterrà, separati da una linea vuota, i cognomi dei soci da ricercare. Nel caso di soci con lo stesso cognome il programma cancellerà tutti i soci con quel cognome.

Nel main dovranno quindi essere svolte le seguenti operazioni:

- inserimento dei soci nella struttura dati
- stampa di tutti i soci dell'associazione: la stampa dovrà avvenire sul file Stampa1Es1.txt
- ricerca dei soci e stampa dei loro dati nel file OutputRicercaEs1.txt
- cancellazione dei soci e stampa dei soci rimanenti nel file Stampa2Es1.txt

I nomi dei file utilizzati dovranno essere inseriti come parametri a riga di comando nel seguente ordine:

InputEs1.txt Stampa1Es1.txt RicercaEs1.txt OutputRicercaEs1.txt
CancellazioneEs1.txt stampa2Es1.txt

Esercizio 2

Si scriva un programma che data un'espressione matematica in notazione RPN (Reverse Polish Notation o notazione polacca inversa) la valuti utilizzando una pila.

Note implementative

La notazione che normalmente si usa per rappresentare le espressioni matematiche è chiamata Infissa. Nella notazione infissa l'operatore è posto tra le due espressioni matematiche che rappresentano i suoi operandi. Per evitare l'ambiguità e far sì che l'espressione possa essere interpretata correttamente si usano le parentesi e le regole di precedenza.

Esistono anche notazioni alternative per rappresentare le espressioni aritmetiche. Tra queste, quella forse più rilevante è la notazione polacca inversa (o RPN). In questa notazione l'operando segue gli operatori. Di seguito è presentato qualche esempio di espressione rappresentata sia in notazione infissa che in RPN:

a + b -> a b +
a + b * c -> a b c * +
(a + b) * c -> a b + c *
(9 + 7) * (10 - (3 + 1)) -> 9 7 + 10 3 1 + - *

Come si può notare, nella RPN per stabilire la priorità tra gli operatori non sono necessarie né parentesi né regole di precedenza.
Inoltre un'espressione in RPN risulta essere più semplice da valutare via software

rispetto alla stessa espressione rappresentata in notazione infissa.

Per valutare un'espressione in RPN si usa una pila e si scandisce l'espressione da sinistra a destra. L'algoritmo è il seguente (si supponga che ogni operatore abbia arietà pari a 2):
1. Si legge il prossimo termine dalla sequenza
2. Se è un operando, lo si inserisce nello stack
3. Se è un operatore:
    - si estraggono dallo stack i due operandi
    - si esegue l'operazione
    - si inserisce il risultato nello stack.
4. Se la sequenza non è vuota si torna al passo 1.

Al termine della valutazione, nello stack c'è un solo valore che rappresenta il risultato dell'espressione.

Si considerino solo le operazioni di addizione, sottrazione, moltiplicazione, divisione tra interi (ovvero si consideri solo la parte intera del risultato e si ignori il resto).
Il risultato dell'espressione valutata potrebbe essere un numero negativo.

Il file che contiene il main si dovrà chiamare Esercizio2.java.
Le espressioni in RPN da valutare dovranno essere lette dal file InputEs2.txt (una riga per ogni espressione, operandi e operatori separati da uno spazio).
I risultati delle espressioni dovranno essere stampati nel file OutputEs2.txt (un risultato per riga).

I nomi dei file utilizzati dovranno essere inseriti come parametri a riga di comando nel seguente ordine:

InputEs2.txt OutputEs2.txt

Esercizio 3

Si scriva un programma che costruisca un albero binario quasi completo e lo stampi in maniera "indentata" (effettuando una previsita).

Note implementative

Un albero binario si dice quasi completo se tutti i livelli, tranne al più l'ultimo sono completi e le foglie dell'ultimo livello sono tutte addossate a sinistra.

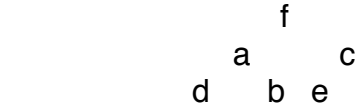
I valori da inserire nell'albero dovranno essere letti dal file InputEs3.txt nel quale saranno elencati separati da uno spazio. Nel file potrebbero essere presenti più righe, ogni riga rappresenta un albero diverso.
L'elemento in posizione *i* avrà il figlio sinistro in posizione *i*2* e il figlio destro in posizione *i*2+1*. La radice si trova in posizione 1.

Esempio:

La riga

f a c d b e

rappresenta l'albero:



che dovrà essere stampato:

```
f
   a
      d
      b
   c
      e
```

Il file che contiene il main si dovrà chiamare Esercizio3.java. Gli alberi dovranno essere stampati nel file OutputEs3.txt; ogni livello di indentazione deve essere costituito da 5 spazi. Ogni albero dovrà essere separato dal successivo da due linee bianche.

I nomi dei file utilizzati dovranno essere inseriti come parametri a riga di comando nel seguente ordine:

InputEs3.txt OutputEs3.txt

Esercizio 4

Implementare un albero Red-Black in cui ogni nodo è memorizzata anche la sua b-altezza (o altezza nera).

Implementare l'operazione di inserimento di un nuovo nodo in modo tale che questo campo, dopo ogni inserimento, risulti essere aggiornato in modo corretto.

Implementare anche un metodo che realizzi l'invisita dell'albero e che stampi per ogni nodo il valore del nodo, uno spazio e tra parentesi il suo colore, suo padre, la sua altezza secondo il formato rappresentato da questo esempio:
80 (RED-F:85-B:2).
I nodi dovranno essere stampati uno per ogni linea in linee consecutive (cioè una sotto all'altra).

Stimare la complessità dell'operazione di inserimento implementata.

Note implementative

I valori da inserire nell'albero dovranno essere letti dal file InputEs4.txt nel quale saranno elencati separati da uno spazio. Si può supporre si tratti di numeri interi. I valori devono essere inseriti nell'albero nell'ordine nel quale sono incontrati leggendo il file. Nel file potrebbero essere presenti più righe, ogni riga rappresenta

un albero diverso.

Il file che contiene il main si dovrà chiamare Esercizio4.java. Gli alberi dovranno essere stampati nel file OutputEs4.txt; ogni albero dovrà essere separato dal successivo da due linee bianche.

I nomi dei file utilizzati dovranno essere inseriti come parametri a riga di comando nel seguente ordine:

InputEs4.txt OutputEs4.txt

Esercizio 5

Scrivere un programma che implementi le operazioni di inserimento, visita e ricerca in un B-tree.
Si supponga che il B-tree contenga solo chiavi e che le chiavi all'interno di ogni nodo siano memorizzate in un array.

Dovranno essere fornite due diverse implementazione dell'operazione di visita. La prima sarà una previsita e il risultato dovrà essere stampato in maniera indentata (si veda esercizio 3). La seconda sarà una visita effettuata in maniera tale che l'output risultante sia costituito da una stampa di tutti i valori presenti nell'albero in ordine crescente.

Dovranno essere fornite due diverse implementazione dell'operazione di ricerca. Nella prima implementazione l'array di ogni nodo deve essere scorso in maniera sequenziale, mentre nella seconda implementazione deve essere effettuata una ricerca dicotomica (o binaria) sull'array.

In entrambe le implementazioni, ponendo t=10, occorre contare il numero complessivo di accessi agli array di tutti i nodi nei quali è stata effettuata la ricerca.
Occorrerà poi produrre quattro istogrammi nei quali si rappresentano il numero di accessi agli array effettuati nei casi di ricerca lineare e binaria avvenute con successo e insuccesso.
Per ogni istogramma sull'asse delle x si rappresentano il numero di accessi, mentre sull'asse delle y il numero di ricerche. Ogni colonna dell'istogramma rappresenta il numero di ricerche che ha prodotto quel numero di accessi. Si richiede di commentare opportunamente i risultati mostrati dai grafici.

Note implementative

Il BTree implementato dovrà funzionare con qualsiasi valore di t (purché > 1).

Per produrre gli output richiesti si ponga t=10.

Le chiavi da inserire nell'albero dovranno essere lette dal file InputEs5.txt nel quale sono elencate una per ogni riga. Le chiavi devono essere inserite nell'albero nell'ordine nel quale sono incontrate leggendo il file.

L'output della previsita indentata dovrà essere stampato nel file Output3Es5.txt;

ogni livello di indentazione dovrà essere costituito da 5 spazi, i valori all'interno di ogni nodo dovranno essere stampati separati da uno spazio.
L'output della seconda visita dovrà essere stampato nel file Output4Es5.txt; un valore per ogni riga.

Le chiavi da ricercare dovranno essere lette dal file RicercaEs5.txt. Il risultato di ogni operazione di ricerca con modalità dicotomica dovrà essere stampato nel file Output1Es5.txt: 1 se l'elemento è stato trovato, 0 altrimenti. Analogamente, il risultato di ogni operazione di ricerca con modalità sequenziale dovrà essere stampato nel file Output2Es5.txt. Ovviamente i due file dovranno essere uguali.
I risultati dovranno essere stampati uno per ogni riga.
Il file che contiene il main si dovrà chiamare Esercizio5.java.

I nomi dei file utilizzati dovranno essere inseriti come parametri a riga di comando nel seguente ordine:

InputEs5.txt RicercaEs5.txt Output1Es5.txt Output2Es5.txt Output3Es5.txt Output4Es5.txt

Tempi e modalità di consegna

I file di input verranno pubblicati al seguente link: http://www.cs.unibo.it/~turrini/DIDATTICA/ALGORITMI/ANNO1314/PROGETTO/
entro il 24 dicembre 2013.

Il progetto dovrà essere svolto singolarmente.

Si precisa che verrà anche valutato l'utilizzo corretto e opportuno del linguaggio Java. I file sorgenti dovranno essere opportunamente commentati. E' possibile utilizzare le classi della libreria standard di Java; tuttavia quando è esplicitamente richiesto l'uso di array non è possibile usare Vector, ArrayList o altre strutture dati.

Inoltre è richiesta la redazione di una relazione che dovrà contenere:
- la descrivere delle scelte progettuali attuate
- la stima della complessità computazionale ove richiesto
- i grafici opportunamente commentati dell'esercizio 5.

I sorgenti degli esercizi (file .java) e la documentazione dovrà essere posta in una directory chiamata con il cognome dello studente che ha svolto il progetto. Tale directory andrà poi compressa (formato zip o jar) e spedita a elisa.turrini7@unibo.it entro il 24 gennaio 2014. Per la spedizione è obbligatorio utilizzare l'indirizzo di email istituzionale.
Entro il 10 febbraio 2014 verrà reso noto chi è stato ammesso alla discussione del progetto, discussione che si terrà a partire dal giorno successivo.

Si ricorda che è possibile consegnare gli esercizi solo qualora questi siano funzionanti, ovvero rispettino le specifiche proposte. Per automatizzare la

compilazione, le classi non dovranno essere contenute in pacchetti o suddivise per directory. La compilazione e l'esecuzione avverrà digitando da shell i seguenti comandi:

```
javac *.java

java Esercizio1 InputEs1.txt Stampa1Es1.txt RicercaEs1.txt
OutputRicercaEs1.txt CancellazioneEs1.txt stampa2Es1.txt

java Esercizio2 InputEs2.txt OutputEs2.txt

java Esercizio3 InputEs3.txt OutputEs3.txt

java Esercizio4 InputEs4.txt OutputEs4.txt

java Esercizio5 InputEs5.txt RicercaEs5.txt Output1Es5.txt
Output2Es5.txt Output3Es5.txt Output4Es5.txt
```

Inoltre, per verificare il funzionamento dei programmi verrà utilizzato un programma di test contenuto nel file Tester.jar e che è presente al seguente link:
http://www.cs.unibo.it/~turrini/DIDATTICA/ALGORITMI/ANNO1314/PROGETTO/

Occorre salvare il file Tester.jar nella directory dove sono presenti i file di output. Per eseguirlo si digita il seguente comando:

```
java -jar Tester.jar
```

Si prega quindi, prima di consegnare, di assicurarsi che i programmi possano essere compilati ed eseguiti con i comandi visti sopra e che l'esecuzione del programma di test dia esito positivo. Si tenga presente che il programma di test controlla che l'output prodotto dai vostri programmi (eseguiti con i file di input messi a disposizione) sia corretto. Di conseguenza ottenere un esito positivo dal programma di test non assicura la correttezza complessiva del progetto.

(N.B. Il programma di test non va consegnato)

Change log

16/12/2013: Specificato per ogni esercizio l'ordine dei parametri a riga di comando
19/12/2013: Esercizio 4
Considerare i valori letti dal file come numeri interi
Esercizio 5
Aggiunto file per la ricerca tra i parametri a riga di comando
08/01/2014: Specificato quali classi è possibile usare.
08/01/2014: Esercizio 5
Richiesta implementazione metodi di visita
10/01/2014: Esercizio 5
Specificato che il BTree implementato dovrà funzionare con qualsiasi valore di T (purché >1)
15/01/2014: Specificate le modalità di consegna relative ai file *.java.
Aggiunte istruzioni per l'utilizzo del programma di test.