Algoritmica – Esame di Laboratorio

07/07/2020

Istruzioni

Risolvete il seguente esercizio prestando particolare attenzione alla formattazione dell'input e dell'output. La correzione avverrà in maniera automatica eseguendo dei tests e confrontando l'output prodotto dalla vostra soluzione con l'output atteso. Si ricorda che è possibile verificare la correttezza del vostro programma su un sottoinsieme dei input/output utilizzati. I file di input e output per i test sono nominati secondo lo schema:
`input0.txt output0.txt`
`input1.txt output1.txt`
`...`
Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Una volta consegnata, la vostra soluzione verrà valutata nel server di consegna utilizzando altri file di test non accessibili. Si ricorda di avvisare i docenti una volta che il server ha accettato una soluzione come corretta.

Suggerimenti

Progettare una soluzione efficiente. Prestare attenzione ad eventuali requisiti in tempo e spazio richiesti dall'esercizio. In ogni caso, valutare la complessità della soluzione proposta e accertarsi che sia *ragionevole*: difficilmente una soluzione con complessità $\Theta(n^3)$ sarà accettata se esiste una soluzione semplice ed efficiente in tempo $\mathcal{O}(n)$.

Abilitare i messaggi di diagnostica del compilatore. Compilare il codice usando le opzioni `-g -Wall` di gcc:

```
gcc -Wall -g soluzione.c -o soluzione
```

risolvere *tutti* gli eventuali *warnings* restituiti dal compilatore, in particolar modo quelli relativi alle funzioni che non restituiscono un valore e ad assegnamenti tra puntatori di tipo diverso.

Provare la propria soluzione in locale. Valutare la correttezza della soluzione sulla propria macchina accertandosi che rispetti gli input/output contenuti nel TestSet. In particolare, si consiglia di provare **tutti** gli input/output contenuti nel TestSet usando le istruzioni nella pagina precedente.

Usare `valgrind`. Nel caso in cui il programma termini in modo anomalo o non calcoli la soluzione corretta, è utile accertarsi che non acceda in modo scorretto alla memoria utilizzando `valgrind` (accertarsi di aver compilato il codice con il flag `-g`):

```
valgrind ./soluzione < input0.txt
```

`valgrind` eseguirà il vostro codice sull'input specificato (in questo caso, il file `input0.txt`), mostrando in output dei messaggi di diagnostica nei casi seguenti:

1. accesso (in lettura o scrittura) ad una zona di memoria non precedente allocata;
2. utilizzo di una variabile non inizializzata precedentemente;
3. presenza al termine dell'esecuzione del programma di zone di memoria allocate con `malloc` ma non liberate con `free` (*memory leak*).

Risolvere *tutti* i problemi ai punti 1. e 2. prima di sottoporre la soluzione al server.

Esercizio

Scrivere un programma che legga da tastiera un intero N e una sequenza di N coppie *chiave* e *valore*. Le N chiavi sono interi positivi, **NON** necessariamente distinti. I valori sono stringhe di lunghezza al più 100 caratteri. Il programma deve inserire uno alla volta, nell'ordine dato, le coppie in una tabella hash T di dimensione $2N$. La risoluzione di eventuali conflitti avviene con liste monodirezionali. Per ciascuna coppia, l'elemento corrispondente nella lista memorizza sia la chiave che il valore. Per l'inserimento si deve utilizzare la funzione:

- $h(x) = (x \quad \% \quad 109) \quad \% \quad 2N$

dove x è la chiave di una coppia.

Nell'inserimento di una coppia con chiave x si deve **sempre** controllare che non sia già presente una coppia avente la stessa chiave x. In questo caso infatti la vecchia coppia deve essere **sostituita** con la nuova.

Al termine dell'inserimento, il programma deve leggere un intero K e stampare i valori delle coppie nella lista $T[K]$, **ordinate** lessicograficamente.

L'input è formattato nel seguente modo. La prima riga contiene l'intero N. Seguono poi $2N$ righe, due righe per coppia. La prima riga della coppia contiene la chiave, mentre la seconda contiene il valore. L'ultima riga dell'input contiene il valore K.

L'output deve riportare i valori delle coppie presenti nella lista $T[K]$, ordinati lessicograficamente e stampati uno per riga. Il programma stampa *vuota* se la K-esima lista è vuota.

Esempio

Input

```
10
109
TyrionLannister
3
NedStark
4
RobertBaratheon
3
KhalDrogo
0
CatelynStark
0
KingJoffrey
110
JonSnow
0
DaenerysTargaryen
1
WinterIsComing
6
Hodor
0
```

Output

```
DaenerysTargaryen
TyrionLannister
```