Algoritmica – Prova di Laboratorio

Corso A e B

Appello del 25/01/2017

Istruzioni

Risolvete il seguente esercizio prestando particolare attenzione alla formattazione dell'input e dell'output. La correzione avverrà in maniera automatica eseguendo dei test 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 $\Theta\left(n^3\right)$ 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 **tutti** gli input/output contenuti nel TestSet. Al fine di agevolare la verifica, si consiglia di posizionare la soluzione compilata e i file appartenenti al TestSet nella stessa directory e lanciare sotto tale directory lo script seguente:

```
for i in input* ; do
  ./soluzione < ${i} | diff -q - output${i##input}
done
```

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`:

```
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

Il programma deve leggere N stringhe, ognuna di lunghezza al più 100 caratteri. Le stringhe devono essere inserite in un albero di ricerca **non** bilanciato utilizzando l'ordinamento lessicografico. Per l'inserimento delle chiavi nell'albero deve essere rispettato il loro ordine nella sequenza.

Data una chiave u, sia $m(u)$ la chiave più piccola presente nel sottoalbero radicato nel nodo con chiave u. Siano u e v due stringhe di lunghezza rispettivamente $|u|$ e $|v|$. Si ricorda che una stringa u è *prefisso* di una stringa v se entrambe le seguenti condizioni sono soddisfatte:

1. la stringa u non è più lunga di v (cioè $|u| \leq |v|$);
2. la stringa u è uguale ai primi $|u|$ caratteri di v.

Ad esempio, "`Ciao`" è prefisso di "`Ciao mondo`". Si noti in particolare che, secondo la definizione, ogni stringa è prefissa di sé stessa.

Si scriva dunque quindi una funzione `prefix` che, dato in input l'albero costruito al punto precedente, stampi tutte le chiavi u per cui $m(u)$ è un prefisso di u. Le chiavi devono essere stampate in ordine lessicografico.

NOTA IMPORTANTE La complessità della funzione `prefix` deve essere *lineare*. Inoltre, **non** saranno considerate valide le soluzioni che memorizzano informazioni aggiuntive nei nodi dell'albero quali, ad esempio, la chiave più piccola che si trova nel sottoalbero radicato in ciascun nodo. Ogni nodo deve contenere solo la sua chiave e i puntatori ai suoi figli.

L'**input** è formattato nel seguente modo. La prima riga contiene l'intero N. Seguono poi N righe, una riga per ogni chiave da inserire nell'albero. Ogni chiave è lunga al più 100 caratteri.

L'**output** è costituito dalle chiavi u per cui $m(u)$ è prefisso di u, una riga per chiave, in ordine lessicografico.

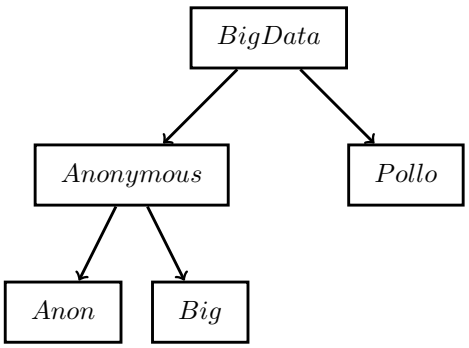
Esempio

Nota: il testo in verde è da intendersi come *commento* e dunque non fa parte dell'input/output.

Input

```
5 /* Numero di chiavi */
BigData
Anonymous
Pollo
Anon
Big
```

Albero



Output

```
Anon      /* m(Anon) = Anon */
Anonymous /* m(Anonymous) = Anon */
Big       /* m(Big) = Big */
Pollo     /* m(Pollo) = Pollo */
/* BigData non presente perché m(BigData) = Anon */
```