Algoritmi e Strutture Dati – Prova di Laboratorio

27/06/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valuato dai docenti in termini di complessità.

Nota Bene: La consegna del compito conclude la prova.

Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. 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.

Esercizio

Si consideri un sistema per la gestione di una rete informatica composta da N nodi. Ciascun nodo è caratterizzato da un ID intero e positivo, e da un tipo tra *Server*, *Client*, *Filtro* e *Router*, rappresentati dai caratteri S, C, F e R rispettivamente. I nodi della rete sono memorizzati tramite un albero binario di ricerca (ABR) usando l'ID come etichetta.

Un cammino che raggiunge un *Client* si dice *completo* se ha origine da un nodo *Server*, attraversa **almeno** un nodo *Filtro* e **nessun altro** *Server*.

Per ciascun *Server*, si definisce il numero g di *Clienti* serviti come il numero di cammini completi che hanno origine dal server stesso.

Scrivere un programma che consideri i *Server* in ordine di ID non decrescente, e stampi per ciascuno di essi il suo ID e il numero di *Client* da lui serviti. (complessità al più $\mathcal{O}(n)$).

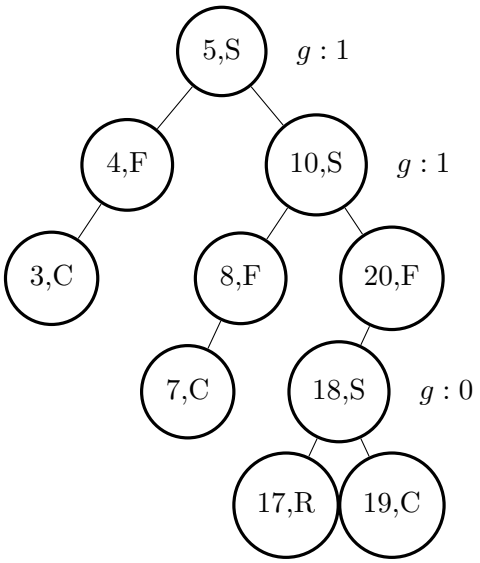
L'**input** è formattato nel seguente modo: la prima riga contiene l'intero N. Seguono N righe contenenti una coppia $ID, tipo$ ciascuna, con i valori separati da uno spazio.

L'**output** contiene una coppia ID, g ciascuna, con i valori separati da uno spazio.

Esempio

Input

```
10
5 S
4 F
10 S
3 C
8 F
20 S
7 C
18 S
19 C
17 R
```



Output

```
5 1
10 1
18 0
```