Algoritmi e Strutture Dati – Prova di Laboratorio

11/04/2025

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. 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 che memorizza informazioni relative al numero di passeggeri saliti e scesi dal treno alle varie fermate. Ogni fermata è caratterizzata da una stringa *nome*, da un intero *passeggeri_scesi* e da un intero *passeggeri_saliti*. Le fermate vengono memorizzate in un albero binario di ricerca (ABR) aventi come etichette delle stringhe, in cui ogni nodo dell'albero é una fermata il cui *nome* ne rappresenta l'etichetta. All'interno dell'albero, il campo $passeggeri\_scesi(u)$ rappresenta il numero di passeggeri scesi alla fermata u, mentre $passeggeri\_saliti(u)$ rappresenta il numero di passeggeri saliti alla fermata u. Sia $parent(u)$ il padre del nodo u. Si definisce $C(u)$ il numero di passeggeri presenti a bordo del treno al nodo u, ed è calcolato come $C(u) = C(parent(u)) + (passeggeri\_saliti(u) - passeggeri\_scesi(u))$. Si noti che la radice dell'albero avrà $C(u) = passeggeri\_saliti$. Ogni percorso dalla radice ad una foglia rappresenta un possibile percorso del treno.

Si scriva un programma che

- legga da tastiera N triple $\{stringa, intero, intero\}$, ciascuna rappresentante rispettivamente, il nome, il numero di passeggeri saliti e il numero di passeggeri scesi ad una fermata, e le inserisca all'interno dell'ABR usando il valore *nome* come etichetta. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette $\leq$ vanno inserite a sinistra);
- Un nodo u si definisce valido se $C(u) \geq 0$ e tutti i suoi discendenti sono nodi validi;
- Si stampino i nomi dei nodi validi in ordine non decrescente.

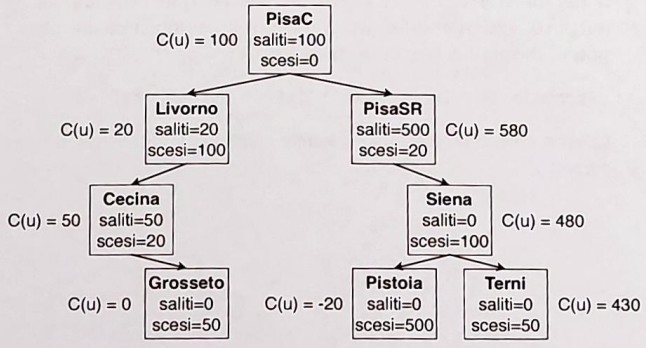
L'**input** è formattato nel seguente modo: la prima riga contiene l'intero N. Seguono N righe contenenti una tripla $\{stringa, intero, intero\}$ ciascuna. La stringa rappresenta il *nome* della fermata, il primo intero *passeggeri_saliti* e il secondo intero *passeggeri_scesi*.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
8
PisaC 100 0
Livorno 20 100
Cecina 50 20
Grosseto 0 50
PisaSR 500 20
Siena 0 100
Terni 0 50
Pistoia 0 500
```



Output

```
Cecina
Grosseto
Livorno
Terni
```