Algoritmi e Strutture Dati – Prova di Laboratorio

19/07/2018

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 le informazioni relative a dei lavoratori. Ciascun lavoratore è identificato da un **ID** univoco e intero, e da un **Cognome** di tipo stringa. Il sistema mantiene questi dati dentro una tabella hash con liste di trabocco (metodo di concatenazione). Scrivere un programma che

- legga da tastiera una sequenza di N coppie (**ID**,**Cognome**) ciascuna rappresentante un lavoratore;
- salvi dentro una *tabella Hash* le informazioni relative ai lavoratori, utilizzando la seguente funzione hash:

  $$h(ID) = \{[(\ a \times ID\ ) +\ b\ ]\ \%\ p\ \}\ \%\ 2N$$

  dove p=999149 , a=1000 e b=2000;
- identifichi i primi K indirizzi della *tabella Hash* in ordine decrescente di collisioni generate. A parità di numero di collisioni considerare l'indirizzo minore (complessità al più $\mathcal{O}(nlogn)$);
- dati gli indirizzi ottenuti e ordinati al passo precedente, considerarli nello stesso ordine e stampare per ciascuno di essi l'**ID** del primo elemento secondo l'ordine lessicografico per **Cognome**. A parità di **Cognome**, scegliere l'elemento con **ID** più basso. In caso di lista vuota, stampare -1(complessità al più $\mathcal{O}(n)$).

L'**input** è formattato nel seguente modo: le prime due righe contengono gli interi N e K. Seguono N righe contenenti una coppia (**intero**,**stringa**) ciascuna.

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

Esempio

Input

```
5 5
1111 Green
2222 Fontana
5555 Lupo
1456 Bernard
777 McCoy
```

Output

```
1456
777
2222
5555
-1
```