Algoritmica – Prova di Laboratorio

Corso A e B

Appello del 04/07/2016

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

Il programma deve leggere una sequenza di $n \geq 2$ interi ed inserirli in un **albero binario di ricerca non bilanciato**. Gli interi sono tutti positivi, con possibili duplicati. Si utilizzi la seguente `struct` per definire i nodi dell'albero:

```
typedef struct n {
    int key;
    struct n* left;
    struct n* right;
} node;
```

Al termine dell'inserimento, il programma deve leggere due interi, `x` e `y`, e stampare il loro *Lowest Common Ancestor* (`lca`), i.e., la chiave del più profondo nodo dell'albero da cui discendono i cammini contenenti `x` e `y`. Ogni nodo è inteso come il discendente di sé stesso. A tal fine, si implementi la seguente funzione:

```
int lca(node* root, int x, int y);
```

Note:

- Si assuma che `x` e `y` siano presenti nell'albero, quindi che il loro `lca` esista sempre.
- Al fine del superamento della prova, la visita deve richiedere tempo al più **lineare rispetto all'altezza dell'albero**.

Suggerimenti:

- Si faccia uso della **proprietà di ricerca** degli alberi binari di ricerca.
- Osservare attentamente gli esempi forniti nelle pagine successive può essere utile nel determinare la corretta strategia implementativa.

L'input è formattato nel seguente modo. La prima riga contiene il numero n di interi da inserire nell'albero. Seguono poi n righe, una per ogni intero nella sequenza. Infine, le ultime due righe corrispondono ai valori di `x` e `y`.

L'output è costituito da una singola riga contenente `lca(x, y)` e il **carattere di new-line** `'\n'`.

Esempi

Note:

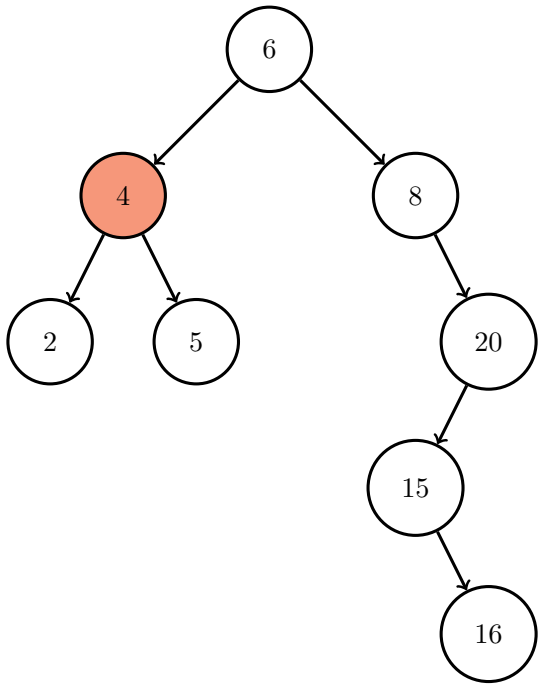
- il testo in verde è da intendersi come *commento* e dunque non fa parte dell'input né dell'output;
- nell'albero finale, il nodo che rappresenta `lca(x, y)` è evidenziato in rosso.

Esempio 1

Input

```
8    // n
6    // primo valore
4    // secondo valore
2    // ...
5
8
20
15
16
2    // x
5    // y
```

Albero finale



Output

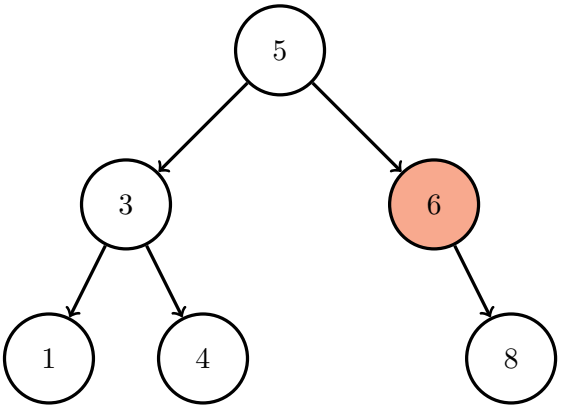
```
4 // lca(2, 5)
```

Esempio 2

Input

```
5    // n
6    // primo valore
8    // secondo valore
3    // ...
1
4
8    // x
6    // y
```

Albero finale



Output

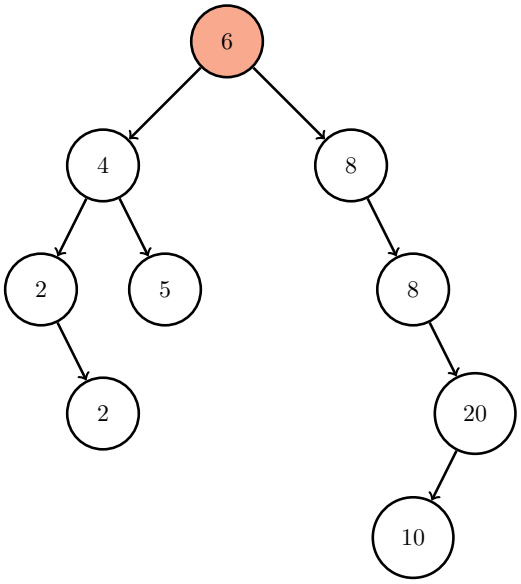
```
6 // lca(8, 6)
```

Esempio 3

Input

```
9    // n
6    // primo valore
4    // secondo valore
2    // ...
8
5
8
20
10
2
5    // x
20   // y
```

Albero finale



Output

```
6 // lca(5, 20)
```