

# Algoritmica – Prova di Laboratorio

Corso A e B

Appello del 09/06/2016

## 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 redirectione 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 *warning* restituiti dal compilatore, in particolare 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 definiti dal TestSet. In particolare, si consiglia di provare **tutti** gli input/output previsti 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**:

```
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 precedentemente 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**. Si utilizzi la seguente struct per definire i nodi dell'albero:

```
typedef struct _node {
    int key;           /* Chiave */
    struct _node *left; /* Sottoalbero sinistro */
    struct _node *right; /* Sottoalbero destro */
} tree_node;
```

Al termine dell'inserimento, stampare il **secondo** elemento più grande nella sequenza in input tramite una opportuna visita sull'albero. **Si assuma che il numero  $n$  di interi in input sia non inferiore a due.**

**Nota:** Al fine del superamento della prova, la visita deve richiedere tempo al più **lineare rispetto all'altezza dell'albero**.

**Suggerimento:** osservare attentamente i due esempi forniti nelle pagine successive può essere utile nel determinare la corretta strategia di visita da implementare.

**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.

**L'output** è costituito da una singola riga contenente l'intero identificato dal programma.

## Esempi

Nota:

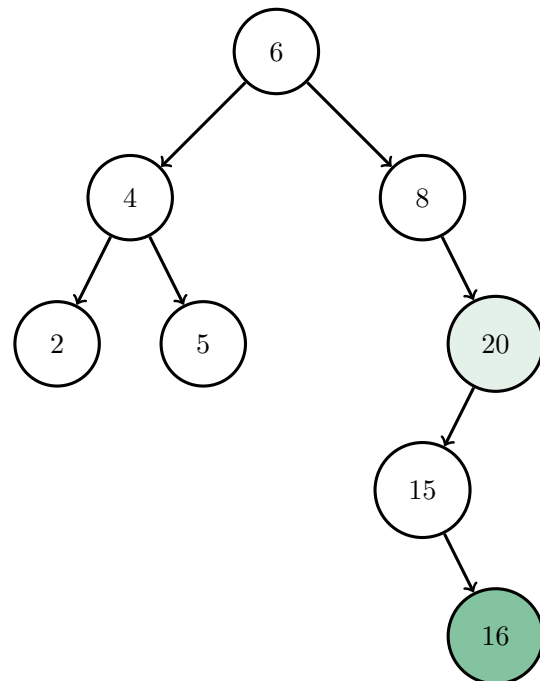
- il testo in verde è da intendersi come *commento* e dunque non fa parte dell'input;
- nell'albero finale, il nodo relativo all'elemento più grande nell'albero è evidenziato in verde chiaro, mentre quello relativo al secondo elemento più grande (cioè l'elemento da restituire) è evidenziato in verde scuro.

### Esempio 1

Input

8    *# N*  
6    *# Primo valore*  
4    *# Secondo valore*  
2    *# etc.*  
5  
8  
20  
15  
16

Albero finale



Output

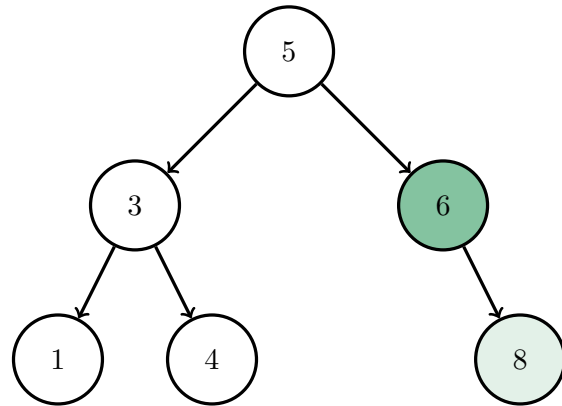
16    *# Secondo elemento più grande*

## Esempio 2

### Input

5     # *N*  
6     # *Primo valore*  
8     # *Secondo valore*  
3     # *etc.*  
1  
4

### Albero finale



### Output

6   # *Secondo elemento più grande*