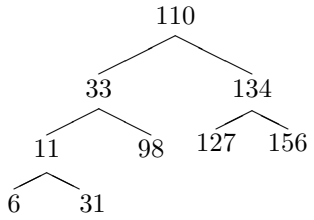
Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 15 Gennaio 2013

Esercizio 3: stampaalbero

Nel materiale didattico predisposto per la penultima lezione frontale, è descritta una funzione `printtree` per la stampa a video di un albero binario (non necessariamente di ricerca).

Ad esempio, l'albero raffigurato qui di seguito:



viene visualizzato dalla funzione *printtree* come:

					110			
			33				134	
	11			98		127		156
6		31						

L'implementazione di una funzione per la stampa di un albero si semplifica *notevolmente* se ci accontentiamo di visualizzare una versione *trasposta* dell'albero stesso, mettendo la radice sulla 0-esima colonna, e in genere un nodo di livello i, nell'i-esima colonna, invece che nell'i-esima riga. Nel nostro esempio si ottiene:

			6
		11	
			31
	33		
		98	
110			
		127	
	134		
		156	

L'osservazione fondamentale al riguardo (cfr. con i lucidi relativi a `printtree`) è che un nodo k dell'albero deve essere visualizzato:

- Nella *colonna* corrispondente al livello in cui k appare nell'albero.
- Nella *riga* corrispondente alla posizione di k nella visita *in-order* dell'albero.

Si chiede di realizzare una funzione:

`void printsearchtree(searchtree *p)`

che avendo in input la radice `p` di un `searchtree` (implementato tramite il codice contenuto nei file `searchtree.c`, `searchtree.h`, disponibili nel materiale predisposto per questa lezione), stampi a video, in forma trasposta, il contenuto dell'albero.

Il programma di test deve poter essere lanciato come:

`stampaalbero` n $file$

dove n è un intero positivo e $file$ è il nome di un file contenente almeno n interi (ad esempio provare il programma con i file `numeri0.txt` e `numeri.txt`).

Il programma deve:

- leggere n interi da $file$;
- inserire gli interi letti in un albero binario di ricerca nello stesso ordine in cui vengono letti da $file$;
- stampare a video l'albero così ottenuto.

Assumiamo inoltre che $file$ contenga solo interi non-negativi di al più 3 cifre. Dunque il valore intero di un nodo può convenientemente essere visualizzato tramite la specifica `%3d` di `printf`, in modo da ottenere, per l'albero di esempio, l'output:

```
               [  6]
          [ 11]
               [ 31]
     [ 33]
          [ 98]
[110]
          [127]
     [134]
          [156]
```

Supplemento

Che modifiche si devono apportare per visualizzare l'albero in modo tale che un figlio sinistro occorra in una riga sottostante a quella dove occorre il padre? Nell'albero di esempio, come si ottiene il seguente output?

```
          [156]
     [134]
          [127]
[110]
          [ 98]
     [ 33]
               [ 31]
          [ 11]
               [  6]
```

In che ordine devo inserire gli elementi per ottenere l'albero del nostro esempio? (N.B. gli elementi sono inseriti in un albero binario di ricerca, dunque tale albero in genere non si ottiene utilizzando la funzione

`buildtree` vista a lezione, che costruisce un albero binario —in genere non di ricerca— a partire da un array.)