15.4 Alberi binari in ricerca

L'algoritmo di ricerca che abbiamo implementato non è dei più veloci. Infatti, per accorgersi che un valore non è presente si deve visitare l'intero albero, realizzando cioè quella che abbiamo definito nel Capitolo 5 come una ricerca completa. Osserviamo però che, preso un qualsiasi nodo, il suo valore è maggiore di tutti i nodi presenti nel suo sottoalbero sinistro e minore di tutti i nodi del suo sottoalbero destro. Una ricerca che utilizzi questa informazione porta più velocemente al risultato (si veda il Listato 15.3).

Se il valore ricercato non è quello presente nel nodo attuale lo andiamo a ricercare nel sottoalbero sinistro se risulta esserne maggiore, altrimenti nel suo sottoalbero destro. Considerando l'albero di Figura 15.1 la ricerca del valore 23 o 200 si conclude in tre chiamate; se ricerchiamo un valore non presente la ricerca si chiude al massimo in cinque chiamate (il numero di livelli più uno).
In pratica abbiamo realizzato un algoritmo che ha prestazioni analoghe a quello di ricerca binaria. L'idea che sta dietro a questo modo di procedere è utilizzata per creare gli indici dei file.

```
/* Ricerca ottimizzata */

void ric_bin(struct nodo *p, int val, struct nodo **p_ele)
{
if(p!=NULL)
  if(val == p->inf) {
    printf("  trovato ");
    *p_ele = p;
  }
else
  if(val < p->inf) {
    printf("  sinistra");
    ric_bin(p->alb_sin, val, p_ele);
  }
  else {
    printf(" destra");
    ric_bin(p->alb_des, val, p_ele);
  }
}
```

Listato 15.3 Ricerca di un valore nell'albero binario

✓ **NOTA**

Osserviamo che la ricerca ha le stesse prestazioni di quella binaria solamente se l'albero è bilanciato, cioè se, preso un qualsiasi nodo, il numero di nodi dei suoi due sottoalberi differisce al più di una unità. Il numero di livelli dell'albero, infatti, costituisce un limite superiore al numero di accessi effettuati dalla ricerca.
La struttura dell'albero creato con la funzione `crea_nodo` dipende dalla sequenza con cui vengono immessi i valori in ingresso; il caso peggiore si presenta quando questi sono già ordinati. Per esempio, se l'utente inserisse: 23, 32, 44, 104, 121, 152, 200 l'albero si presenterebbe totalmente sbilanciato, un albero cioè dove il numero di livelli è uguale al numero di elementi inseriti. Altre sequenze danno vita ad alberi più bilanciati.

Si potrebbero scrivere algoritmi che bilanciano l'albero mentre lo creano, ma spesso nella realtà – specialmente se si sta lavorando con la memoria secondaria – questo risulta pesante in termini di tempi di risposta. Un'altra soluzione è quella di prevedere funzioni che trasformino alberi qualsiasi in alberi bilanciati, per cui l'inserimento di elementi avviene così come l'abbiamo visto e, a tempi determinati, viene lanciata la procedura di bilanciamento.