## 15.2 Implementazione di alberi binari

Consideriamo ora il seguente problema: memorizzare i dati interi immessi dall'utente in un albero binario tale che il valore dell'etichetta di un qualsiasi nodo sia maggiore di tutti i valori presenti nel suo sottoalbero sinistro e minore di tutti i valori del suo sottoalbero destro. Per esempio, se l'utente immette in sequenza 104, 32, 44, 121, 200 152, 23, un albero conforme con la definizione è quello di Figura 15.1. La sequenza di immissione termina quando l'utente inserisce il valore zero; in caso di immissione di più occorrenze dello stesso valore, soltanto la prima verrà inserita nell'albero. Si chiede di visitare l'albero in ordine anticipato.

Una soluzione è ottenuta definendo ciascun nodo come una struttura costituita da un campo informazione, contenente l'etichetta, e due puntatori al sottoalbero sinistro e al sottoalbero destro:

```
struct nodo {
       int inf;
       struct nodo *alb_sin;
       struct nodo *alb_des;
     };
```

L'albero binario di Figura 15.1 verrebbe così memorizzato come in Figura 15.2. I puntatori che non fanno riferimento a nessun nodo devono essere messi a valore `NULL`, in modo da permettere una corretta terminazione delle visite. Il programma completo è presentato nel Listato 15.1.

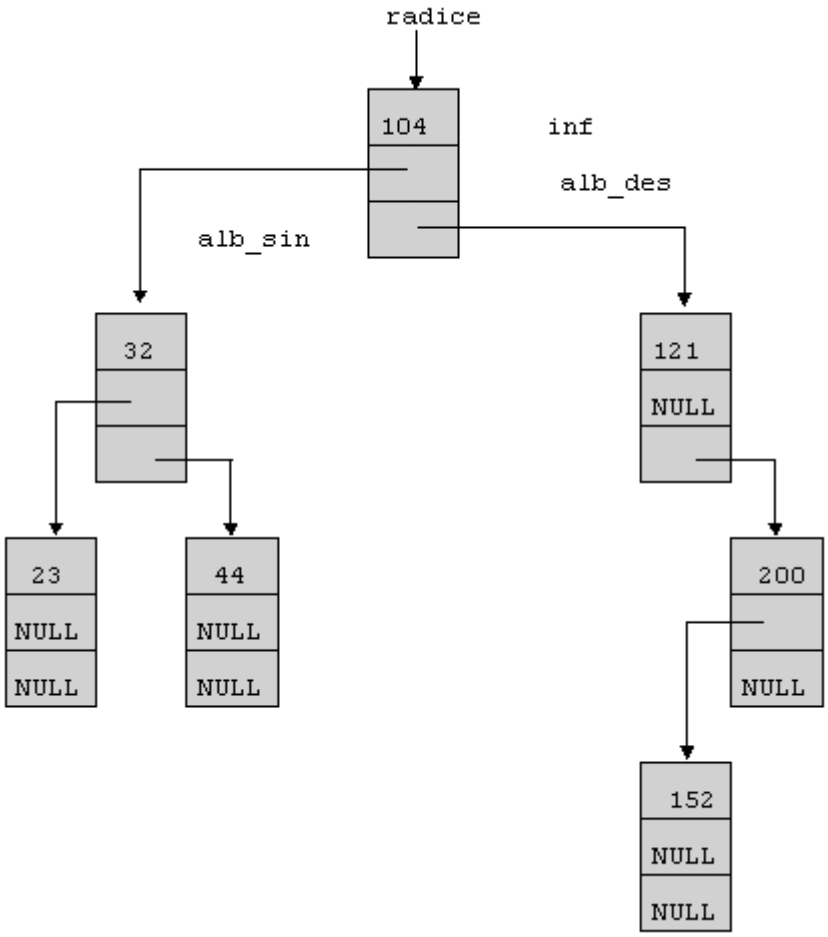


Figura 15.2 Esempio di memorizzazione dell'albero binario di Figura 15.1 in una lista doppia

Nel `main` è dichiarata la variabile `radice` che conterrà il riferimento alla radice dell'albero; essa è un puntatore a oggetti di tipo `nodo`:

```
struct nodo *radice;
```

Il `main` invoca la funzione `alb_bin` che crea l'albero e ritorna il puntatore alla radice:

```
radice = alb_bin();
```

Successivamente chiama la funzione di visita, che visualizza le etichette in ordine anticipato:

```
anticipato(radice);
```

Ad `anticipato` viene passato il puntatore alla radice dell'albero.

Listato 15.1 - Creazione e visita in ordine anticipato di una albero binario

La funzione `alb_bin` crea l'albero vuoto inizializzando a `NULL` il puntatore alla radice `p`. Il resto della procedura è costituito da un ciclo in cui si richiedono all'utente i valori della sequenza finché non viene immesso zero. A ogni nuovo inserimento viene chiamata la funzione `crea_nodo` la quale pensa a inserirlo nell'albero:

```
p = crea_nodo(p, x.inf);
```

A ogni chiamata `crea_nodo` restituisce la radice dell'albero stesso. La dichiarazione di `crea_nodo` è la seguente:

```
struct nodo *crea_nodo(struct nodo *p, int val);
```

Come abbiamo già visto per le liste, il caso della creazione del primo nodo deve essere trattato a parte. La funzione `alb_bin` ha provveduto a inizializzare la radice a valore `NULL`; in questo modo un test su `p` in `crea_nodo` dirà se

l'albero è vuoto. In questo caso verrà creato il nodo radice e inizializzato il campo `inf` con il valore immesso dall'utente, passato alla funzione nella variabile `val`:

```
/* Creazione del nodo */
     p = (struct nodo *) malloc(sizeof(struct nodo));
     p->inf = val;
     p->alb_sin = NULL;
     p->alb_des = NULL;
```

Prima di far ritornare il controllo ad `alb_bin` viene assegnato `NULL` ai puntatori sinistro e destro del nodo.
Nel caso esista almeno un nodo già costruito (`p` non è uguale a `NULL`), ci si domanda se `val` sia maggiore del campo `inf` della radice, nel qual caso viene richiamata ricorsivamente `crea_nodo` passandole il puntatore al sottoalbero destro:

```
p->alb_des = crea_nodo(p->alb_des, val);
```

Se al contrario `val` è minore di `inf`, allora viene richiamata ricorsivamente la funzione sul sottoalbero sinistro:

```
p->alb_sin = crea_nodo( p->alb_sin, val);
```

Se nessuno dei due casi risulta vero, significa che `val` è uguale a `inf` e dunque non deve essere inserito nell'albero perché, come specificava il testo del problema, le occorrenze multiple devono essere scartate.
Si noti come, nel caso di un valore non ancora memorizzato nell'albero, il procedimento ricorsivo termini sempre con la creazione di una foglia, corrispondente alle istruzioni di "creazione del nodo" che abbiamo elencato in precedenza. La connessione del nodo creato al padre avviene grazie al valore di ritorno delle chiamate ricorsive, che è assegnato a `p->alb_des` o a `p->alb_sin` secondo il caso.
La funzione `anticipato` corrisponde in maniera speculare alla definizione di visita in ordine anticipato data precedentemente. L'intestazione della definizione della funzione è la seguente:

```
void anticipato(struct nodo *p);
```

Il parametro attuale `p` assume il valore della radice, se non è uguale a `NULL` (cioè se l'albero non è vuoto), ne viene stampata l'etichetta, e viene invocata ricorsivamente `anticipato` passandole la radice del sottoalbero sinistro:

```
anticipato(p->alb_sin);
```

Successivamente viene richiamata la funzione passandole la radice del sottoalbero destro:
anticipato(p->alb_des);