Pseudocodice

F. Alberi binari di ricerca

F.1. Ricerca (versione ricorsiva)

```
Funzione trova (AlberoRicerca r, tipoChiave k) → Nodo
  if r = null then
  |  return null
  else if k < r.chiave then
  |  return trova(r.sx, k)
  else if k > r.chiave then
  |  return trova(r.dx, k)
  else
  └  return r
```

F.2. Ricerca (versione iterativa)

```
Funzione trova (AlberoRicerca r, tipoChiave k) → Nodo
  while r ≠ null and r.chiave ≠ k do
  |  if k < r.chiave then
  |  |  r ← r.sx
  |  else
  |  └  r ← r.dx
  return r
```

Note

- Come nel caso delle liste, supponiamo che nei nodi vi sia un campo *chiave*, utilizzato per le ricerche, rispetto al quale l'albero è ordinato.
- Si noti che i riferimenti vengono passati *per valore*. Per questa ragione nella versione iterativa della funzione *trova* (Funzione F.2) per la ricerca è possibile utilizzare direttamente il parametro r senza necessità di una variabile ausiliaria.
- Come nel caso delle liste, le funzioni di inserimento e cancellazione possono essere trasformate in procedure, se la variabile che contiene il riferimento alla radice dell'albero viene fornita *per riferimento*, anziché per valore.
- Si ipotizza che siano possibili più nodi con la stessa chiave. Nell'inserimento, un nodo con una stessa chiave di un nodo già presente viene inserito nel sottoalbero destro di questo. Nella ricerca viene restituito il primo nodo che viene individuato con la chiave richiesta (se presente). Analogamente per la cancellazione.

F.3. Inserimento (versione ricorsiva)

```
Funzione inserisci (AlberoRicerca r, elemento d) →AlberoRicerca
  /* Inserisce nell'albero di binario di ricerca riferito dal parametro r un
     nuovo nodo contenente il dato di d, nella posizione opportuna.
     Restituisce il riferimento alla radice dell'albero così modificato.     */
  k ← d.chiave
  if r = null then
  |  r ← riferimento a nuovo nodo
  |  r.chiave ← k
  |  r.altri campi← d.altri campi
  |  r.sx ← null
  |  r.dx ← null
  else if k < r.chiave then
  |  r.sx ← inserisci(r.sx, d)
  else
  |  r.dx ← inserisci(r.dx, d)
  return r
```

F.4. Inserimento (versione iterativa)

```
Funzione inserisci (AlberoRicerca r, elemento d) →AlberoRicerca
  /* Inserisce nell'albero di binario di ricerca riferito dal parametro r un
     nuovo nodo contenente il dato di d, nella posizione opportuna.
     Restituisce il riferimento alla radice dell'albero così modificato.     */

  k ← d.chiave
  // Preparazione del nodo da inserire
  t ← riferimento a nuovo nodo
  t.chiave ← k
  t.altri campi← d.altri campi
  t.sx ← null
  t.dx ← null

  // Ricerca la posizione dove inserire
  padre ← null
  n ← r
  while n ≠ null do
  |  padre ← n
  |  if k < n.chiave then
  |  |  n ← n.sx
  |  else
  |  |  n ← n.dx
  // in questo punto padre si riferisce al nodo sotto il quale inserire
  // (contiene null se l'inserimento va fatto alla radice - albero vuoto)

  // Inserisci
  if padre = null then
  |  r ← t
  else if k < padre.chiave then
  |  padre.sx ← t
  else
  |  padre.dx ← t
  return r
```

F.5. Cancellazione

```
Funzione cancella (AlberoRicerca r, tipoChiave k) →AlberoRicerca
  /* Elimina dall'albero di ricerca un nodo di chiave k, se presente.
     Restituisce il riferimento alla radice dell'albero così modificato.     */

  // Ricerca il nodo da cancellare e il suo nodo padre
  padre ← null
  n ← r
  while n ≠ null and n.chiave ≠ k do
      padre ← n
      if k < n.chiave then
          n ← n.sx
      else
          n ← n.dx

  // Cancella il nodo se è stato trovato
  if n ≠ null then
      if n.sx = null then                    // manca figlio sinistro:  sostituisci il
          if padre ≠ null then                   // nodo con il suo sottoalbero destro
              if n.chiave < padre.chiave then
                  padre.sx ← n.dx
              else
                  padre.dx ← n.dx
          else
              r ← r.dx                      // caso particolare:  cancellazione radice
      else if n.dx = null then                // manca figlio destro:  sostituisci il
          if padre ≠ null then                // nodo con il suo sottoalbero sinistro
              if n.chiave < padre.chiave then
                  padre.sx ← n.sx
              else
                  padre.dx ← n.sx
          else
              r ← r.sx                      // caso particolare:  cancellazione radice
      else                                                  // ci sono entrambi i figli
          // ricerca il nodo contenente la chiave più grande
          // del sottoalbero sinistro
          t ← n
          m ← n.sx
          while m.dx ≠ null do
              t ← m
              m ← m.dx
          // in questo punto m si riferisce al nodo contenente la chiave più
          // grande del sottoalbero sinistro, t al padre di m
          n.chiave ← m.chiave           // copia la chiave presente in m nel nodo n
          n.altri campi← m.altri campi          // e copia anche tutti gli altri dati
          // elimina il nodo riferito da m
          if t = n then                // se il nodo m è figlio sinistro del nodo n
              n.sx ← m.sx
          else
              t.dx ← m.sx
  return r
```