ABR con successore

Realizzare una implementazione degli alberi binari di ricerca nella quale il campo *parent* è sostituito dal campo *successore*, ovvero un nodo `x` avrà i campi `x. key`, `x.left`, `x.right` e `x.succ`, mentre non c'è `x.p`.

Soluzione.

Min, Max, Search

Le operazioni di `Min`, `Max`, `Search` restano invariate.

Insert

Per inserire un nodo z, come accade nell'implementazione standard, procedo come in una ricerca, scendendo dalla radice fino ad una foglia, mantenendo, oltre che un riferimento y al potenziale "parent" anche un riferimento *pred* al potenziale predecessore di z (ogni volta che si scende a destra lo si aggiorna, di modo da avere sempre il più prossimo antenato del quale siamo nel sottoalbero destro). Questo serve perché, una volta inserito z, detto `pred` il suo predecessore, il campo successore di `pred` dovrà riferire z, mentre il successore di z sarà quello che prima era il successore di `pred`.

Concretamente, lo pseudocodice può essere il seguente, dove sono commentate solo le "novità" rispetto alla versione ordinaria.

```
Insert(T,z)
   x = T.root
   y = nil
   pred = nil       // pred contiene l'antenato piu' prossimo
                    // del quale si e' nel sottoalbero dx
   while x <> nil
      y = x
      if z.key < x.key
         x = x.left
      else
         x = x.right
         pred = y    // aggiorna il potenziale predecessore

    if y == nil
       T.root = z
    else if z.key < y.key
       y.left = z
    else
       y.right = z

    if pred <> nil         // se ho un predecessore, il successore di z
       z.succ = pred.succ  / diventa quello che era il successore di pred
       pred.succ = z       // mentre il successore di pred sara' z

    else                   // se invece pred=nil, sono sempre sceso a
       z.succ = y          // sinistra quindi z e' il minimo, e il
                           // suo  successore e' il padre
```

La complessità resta $O(h)$ dove h è l'altezza dell'albero.

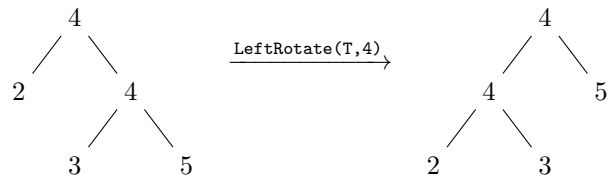
Parent

Per trovare il parent di un nodo x, se assumo che si abbiano ABR nei quali tutte le chiavi siano distinte o nei quali, per ogni nodo w, gli elementi nel sottoablero sinistro hanno chiave $<$ e nel sottoalbero destro $\geq$ di quella di w, posso operare come se stessi cercando la chiave del nodo x, avendo cura di mantenere traccia del parent z del nodo corrente y durante la ricerca.

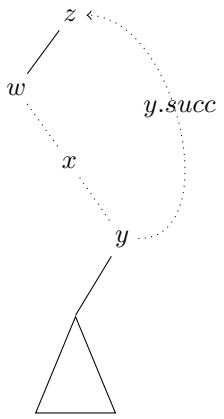
```
Parent(T,x)
   y = T.root       // nodo corrente
   z = nil          // parent del nodo corrente

   while (y <> x)
      z = y
      if x.key < y.key
         y = y.left
      else
         y = y.right

   return z
```

La complessità risulta $O(h)$ dove h è l'altezza dell'albero.

Tuttavia, a meno che le chiavi non siano tutte diverse, l'assunzione che l'albero soddisfi la proprietà menzionata, anche se è stato costruito con inserimenti e cancellazioni, non è ragionevole, dato che viene invalidata dalle rotazioni



Come calcolare dunque il parent in assenza di questa assunzione? L'idea è la seguente: se x non ha sottoalbero destro, il suo successore è il suo antenato più prossimo, per il quale x è nel sottoalbero sinistro, quindi potrò partire da $x.succ$, scendere al figlio sinistro e quindi sempre a destra fino a che incontrerò x, determinando così il parent. Se invece x ha sottoalbero destro, posso fare la stessa cosa, partendo dal massimo del sottoalbero destro. Segue uno schema e quindi lo pseudo-codice:



```
Parent(T,x)
   y = x                 // trovo il massimo del sottoalbero destro di x
   while y.right <> nil  // (se presente)
      y = y.right

   z = y.succ            // il successore di y e' l'antenato piu' prossimo
                         // di x per il quale x sia nel sottoalbero sinistro

   if z <> nil           // quindi da y scendendo a sx e poi sempre a dx
      w = z.left         // incontro x
   else
      w = T.root         // se l'antenato in questione e' nil, z era il max
                         // e quindi x si incontra partendo dalla radice e
                         // andando sempre a dx
   if x == w
      return z

   while w.right <> x
      w = w.right

   return w
```

Nel caso peggiore compio un intero percorso dal nodo x fino ad una foglia e dalla radice fino ad x, quindi la complessità è ancora limitata dall'altterzza dell'albero $O(h)$.

Predecessore

Per determinare il predecessore di un nodo, come per il parent, se assumo che si abbiano ABR nei quali tutte le chiavi sono distinte o nei quali nel sottoalbero sinistro trovo solo nodi con chiave minore della radice, posso operare come se stessi cercando la chiave del nodo x, avendo cura di mantenere traccia del predecessore del nodo corrente. Senza questa assunzione posso usare la procedura ordinaria, nella quale, però, l'accesso al padre non avviene leggendo un campo, quindi in tempo $O(1)$, ma chiamando la procedura `Parent` che ha costo $O(h)$.

```
Pred(T,x)
   if x.left <> nil
      return Max(x.left)
   else
     p = Parent(T,x)
     while (p <> nil and x == p.left)
        x = p
        p = Parent(T,x)

   return p
```

La complessità, dato che posso avere $O(h)$ chiamate di `Parent`, risulta $O(h^2)$.

Cancellazione

Per la cancellazione ho bisogno di una versione diversa della `Transplant` nella quale aggiorno solo il riferimento dal parent al figlio.

```
Transplant(T,u,v)
   p =Parent(T,u)

   if p == nil
      T.root = v
   else
      if (u == p.left)
         p.left = v
      else
         p.right = v
```

Quindi la cancellazione può essere:

```
Delete(T, z)
  pred = Pred(T, z)                // determina il predecessore di z
  y = z.succ                       // e il suo successore

  if (pred <> nil)                 // il predecessore di z dopo la cancellazione
    pred.succ = y                  // ha come succ l'attuale successore di z


  if (z.left == nil)               // la parte rimanente e' analoga a quella std
     Transplant(T, z, z.right)
  else if (z.right == nil)
     Transplant(T, z, z.left)
  else
     if (y <> z.right)
        Transplant(T, y, y.right)
        y.right = z.right

     Transplant(T, z, y)
     y.left = z.left
```

La complessità è data dalla complessità del predecessore, quindi $O(h^2)$.