```
int ABR(TreePtr tPtr)
/* verifica se tPtr è un albero binario di ricerca, di interi,  senza ripetizioni
*postc: restituisce 1 se tPtr e' un albero binario di ricerca, di interi,
*senza ripetizioni, e 0 altrimenti */
{int m = INT_MIN;
return ABRaus(tPtr,&m);}

int ABRaus (TreePtr tPtr, int *prec)
/* verifica se tPtr è un albero binario di ricerca, di interi,  senza ripetizioni
*prec: il valore  di prec e' minore di ogni valore in tPtr.
*postc: restituisce 1 se tPtr e' un albero binario di ricerca, di interi,
*senza ripetizioni, e 0 altrimenti */
{int ris;
if (!tPtr) return 1;
if ( ris = ABRaus(tPtr->left,prec)  && *prec < tPtr->elem)
/* in prec ho sempre l'ultimo nodo visitato, che poiche'
la visita e' inorder deve essere minore di quello correntemente visitato */
{*prec = tPtr->elem;
ris = ris && ABRaus(tPtr->right,prec);}
 return ris;}
/*Perche' prec DEVE essere passato per riferimento?*/


TreePtr Succ(const TreePtr t, int key)
/* restituisce un puntatore al nodo che contiene
*nel campo elem il valore successivo a  key, se presente
prec: t e' un albero binario di ricerca
*postc restituisce il puntatore al successivo di key, se presente,
*il puntatore restituito e' NULL se t e' vuoto, key non é presente oppure
*se il successivo non e' nell'albero perchè key é il massimo  */
{TreePtr tempPtr;
if (!t )  return NULL;
if (key == t ->elem)
    if (t->right)  return trovaMin(t->right); else return tempPtr = NULL;
     /* se il nodo  che contiene key ha un figlio destro il suo successore è il minim
o
    *nel sottoalbero destro */
 if (key < t->elem && t->left)
 {  tempPtr = Succ(t->left,key);
 if (!tempPtr) return t; else return tempPtr;}
/* se il nodo  che contiene key non ha il figlio destro, tempPtr == NULL e
*il successore é t perche' e' il padre del primo antenato  figlio sinistro
  del nodo  che contiene key */
else
 if ( t->right)  return Succ( t->right,key);
 else return NULL;
 }
```