```
(**********************************************************************
**********************************************************************
********
******** TTTtree, a efficient insert/delete data structure
******** Language SML/NY
******** Author M. Bertol (c)
********
******** Release: Thu Aug 15 14:40:26 MET DST 1996
********
**********************************************************************
**********************************************************************)




signature ITEM =
    sig
        type key                                                    key
        val eq : key * key -> bool                                  leq
        val leq : key * key -> bool                                 eq
    end;

signature SETTREE =
     sig
        type tree                                                   tree
        type key                                                    key
        val empty : tree                                            empty
        val insert : key * tree -> tree                             insert
        val delete : key * tree -> tree                             delete
        val lookup : key * tree -> bool                             lookup
    end;


functor TTTree( Item: ITEM ): SETTREE=                              TTTree

struct

    type key = Item.key

    fun leq(p: key, q: key) : bool = Item.leq(p,q);
    fun eq(p: key, q: key) : bool = Item.eq(p,q);

    infix leq;
    infix eq;

    datatype tree = empty
      | tree2 of tree * key * tree * key
      | tree3 of tree * key * tree * key * tree * key
      | leaf of key

    exception   maxEmptyTreeError;

    fun max(empty) = raise maxEmptyTreeError
      | max(leaf(x)) = x
      | max(tree2(_, _, _, x)) = x
      | max(tree3(_, _, _, _, _, x)) = x;


    exception    lookupEmptyTreeError;
```

lookup

```
fun lookup(a: key, empty) = raise lookupEmptyTreeError
  | lookup(a: key, leaf(b)) =  a eq b
  | lookup(a: key, tree2(s,x,t,y)) =
        if (x eq a) orelse (y eq a) then true
        else if a leq x
                then lookup(a, s)
             else if a leq y then lookup(a, t)
                  else false
  | lookup(a: key, tree3(r,x,s,y,t,z)) =
        if  (x eq a) orelse (y eq a) orelse (z eq a) then true
        else if a leq x
                then lookup(a, r)
             else if a leq y then lookup(a, s)
                  else if a leq z then lookup(a, t)
                       else false;

datatype insert_status = grown | proceed;


exception  MisteriousException;

exception   insertEmptyTreeError;

exception   deleteEmptyTreeError;


fun insertA(a: key, empty) = (grown, leaf(a))
  | insertA(a: key, leaf(b)) =
         if a eq b then (proceed, leaf(b))
         else if a leq b
                 then (grown, tree2(leaf(a),a , leaf(b), b))
              else (grown, tree2(leaf(b),b , leaf(a), a))
  | insertA(a: key, tree2(s,x,t,y)) =
         if (x eq a) orelse (y eq a) then (proceed, tree2(s,x,t,y))
         else if a leq x
                 then let val (stat, s') = insertA(a, s)
                      in if stat = grown
                             then case s' of
                                 tree2(s1,x1,t1,y1)
                                 =>  (proceed, tree3(s1,x1,t1,y1,t,y))
                               | tree3(_,_,_,_,_,_)
                                 => raise MisteriousException
                               | empty
                                 => raise insertEmptyTreeError
                               | leaf(_)
                                 => raise MisteriousException
                         else (proceed, tree2(s',max(s'),t,y))
                      end
              else let val (stat, t') = insertA(a, t)
                   in if stat = grown
                         then case t' of
                             tree2(s1,x1,t1,y1)
                             =>  (proceed,  tree3(s,x,s1,x1,t1,y1))
                           | tree3(_,_,_,_,_,_)
                             => raise MisteriousException
                           | empty
                             => raise  insertEmptyTreeError
                           | leaf(_)
                             => raise MisteriousException
                      else (proceed, tree2(s,x,t',max(t')))
```

```
                 end
  | insertA(a: key, tree3(r,x,s,y,t,z)) =
        if  (x eq a) orelse
            (y eq a) orelse
            (z eq a) then (proceed, tree3(r,x,s,y,t,z))
        else if a leq x then let val (stat, r') = insertA(a, r)
                                 val m = max(r')
                             in if stat = grown
                                  then (grown, tree2(r',m,tree2(s,y,t,z),z))
                                else (proceed, tree3(r',m,s,y,t,z))
                             end
             else if a leq y
                     then let val (stat, s') = insertA(a, s)
                          in if stat = grown
                                then case s' of
                                   tree2(s1,x1,t1,y1)
                                   =>  (grown, tree2(tree2(r,x,s1,x1),x1,
                                                 tree2(t1,y1,t,z),z))
                                 | tree3(_,_,_,_,_,_)
                                   => raise MisteriousException
                                 | empty
                                   => raise  insertEmptyTreeError
                                 | leaf(_)
                                   => raise MisteriousException
                             else (proceed, tree3(r,x,s',max(s'),t,z))
                          end
                 else let val (stat, t') = insertA(a, t)
                          val m = max(t')
                      in if stat = grown
                            then (grown, tree2(tree2(r,x,s,y),y,t',m))
                         else (proceed, tree3(r,x,s,y,t',m))
                      end;


fun insert(a: key, t) = let val (_,t') = insertA(a, t) in t' end;          insert

datatype delete_status = propagateDelete | ContainedAndDeleted | notContained;


fun deleteA(a: key, empty) = (notContained, empty)
  | deleteA(a: key, leaf(b)) = if a eq b
                                 then (propagateDelete, empty)
                              else (notContained, leaf(b))
  | deleteA(a: key, tree2(s,x,t,y)) =
        if a leq x then let val (stat, s') = deleteA(a, s)
                        in if stat=propagateDelete
                              then case t of
                                 tree3(r1,x1,s1,y1,t1,z1)
                                 => (ContainedAndDeleted,
                                    tree2(tree2(s',max(s'),r1,x1),x1,
                                          tree2(s1,y1,t1,z1),z1))
                               | tree2(s1,y1,t1,z1)
                                 => (propagateDelete,
                                    tree3(s',max(s'),s1,y1,t1,z1))
                               | leaf(x)
                                 => (propagateDelete, leaf(x))
                               | empty
                                 => raise deleteEmptyTreeError
                           else (stat, tree2(s',max(s'),t,y))
                        end
        else if a leq y then
```

```
          let val (stat, t') = deleteA(a, t)
          in if stat=propagateDelete
                then case s of
                   tree3(r1,x1,s1,y1,t1,z1)
                   => (ContainedAndDeleted,
                       tree2(tree2(r1,x1,s1,y1),y1,
                             tree2(t1,z1,t',max(t')),max(t')))
                 | tree2(s1,y1,t1,z1)
                   => (propagateDelete, tree3(s1,y1,t1,z1,t',max(t')))
                 | leaf(x)
                   => (propagateDelete, leaf(x))
                 | empty
                   => raise deleteEmptyTreeError
            else (stat, tree2(s,x,t',max(t')))
          end
           else (notContained,  tree2(s,x,t,y))
  | deleteA(a: key, tree3(r,x,s,y,t,z)) =
        if a leq x then let val (stat, r') = deleteA(a, r)
                        in if stat=propagateDelete
                              then case s of
                                 tree3(r1,x1,s1,y1,t1,z1)
                                 => (ContainedAndDeleted,
                                     tree3(tree2(r',max(r'),r1,x1),x1,
                                           tree2(s1,y1,t1,z1),z1,
                                           t,z))
                               | tree2(s1,x1,t1,y1)
                                 => (ContainedAndDeleted,
                                     tree2(tree3(r',max(r'),s1,x1,t1,y1),y1,
                                           t,z))
                               | leaf(x1)
                                 => (ContainedAndDeleted,
                                     tree2(s,y,t,z))
                               | empty
                                 => raise deleteEmptyTreeError
                           else (stat, tree3(r',max(r'),s,y,t,z))
                        end
        else if a leq y
                then let val (stat, s') = deleteA(a, s)
                     in if stat=propagateDelete
                           then case r of
                              tree3(r1,x1,s1,y1,t1,z1)
                              => (ContainedAndDeleted,
                                  tree3(tree2(r1,x1,s1,y1),y1,
                                        tree2(t1,z1,s',max(s')),max(s'),
                                        t,z))
                            | tree2(s1,x1,t1,y1)
                              => (ContainedAndDeleted,
                                  tree2(tree3(s1,x1,t1,y1,s',max(s')),max(s'),
                                          t,z))
                            | leaf(x1)
                              => (ContainedAndDeleted,
                                  tree2(r,x,t,z))
                            | empty
                              => raise deleteEmptyTreeError
                         else (stat, tree3(r,x,s',max(s'),t,z))
                     end
             else if a leq z
                     then let val (stat, t') = deleteA(a, t)
                          in if stat=propagateDelete
                                then case s of
                                   tree3(r1,x1,s1,y1,t1,z1)
                                   => (ContainedAndDeleted,
```

```
                                  tree3(r,x,
                                        tree2(r1,x1,s1,y1),y1,
                                        tree2(t1,z1,t',max(t')),max(t')))
                            | tree2(s1,x1,t1,y1)
                              => (ContainedAndDeleted,
                                  tree2(r,x,
                                        tree3(s1,x1,
                                              t1,y1,
                                              t',max(t')),max(t')))
                            | leaf(x1)
                              => (ContainedAndDeleted,
                                  tree2(r,x,s,y))
                            | empty
                              => raise deleteEmptyTreeError
                      else (stat, tree3(r,x,s,y,t',max(t')))
                    end
            else (notContained, tree3(r,x,s,y,t,z));

    fun delete(a,t) = let val (_,t') = deleteA(a,t) in t' end;          delete

end (* TTTree *)                                                        end

structure IntegerItem =                                                 IntegerItem
    struct
        type key = int;                                                 key
        fun leq (i:key, j:key) =  i<=j;                                 leq
        fun eq(i:key, j:key) =  i=j;                                    eq
    end;

structure IntegerTTTree = TTTree(IntegerItem);                          IntegerTTTree
```