// Vistið þessa skrá með nafninu E6.java og klárið að

// forrita föllin searchRecursive, searchLoop og

// searchTailRecursive.  Þið skuluð líka þýða og keyra

// klasann svona (þið þurfið einnig skrána AVL.java):

//    javac AVL.java E6.java

//    java E6

// I forrituninni skuluð þið fylgja þeim stöðulýsingum

// sem gefnar eru.

class E6

{

    public static class Pair<T extends Comparable<? super T>>

    {

        public AVL<T> le,gt;

    }

    // Notkun: searchRecursive(t,x,p);

    // Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.

    //         Ekkert þessara gilda er null og t er í vaxandi

    //         röð.

    // Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.le jafnt null.

    //         p.gt vísar á fremsta hnút í t með gildi >x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.gt jafnt null.

    public static<T extends Comparable<? super T>> void searchRecursive( AVL<T> t, T x, Pair<T> p )

    {

        if( t == null )

        {

            return;

        }

        //note to myself for AVL.rootValue(t).compareTo(x)

        //if t > x, it returns positive number

        //if t < x, it returns negative number

        //if t == x, it returns 0

        else if( AVL.rootValue(t).compareTo(x) <= 0 )

        {

            p.le = t;

            searchRecursive(AVL.right(t), x, p);

        }

        else

        {

            p.gt = t;

            searchRecursive(AVL.left(t), x, p);

        }

    }

    // Notkun: searchTailRecursive(t,x,p);

    // Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.

    //         Ekkert þessara gilda er null og t er í vaxandi

    //         röð.

    // Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.le óbreytt.

    //         p.gt vísar á fremsta hnút í t með gildi >x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.gt óbreytt.

    public static<T extends Comparable<? super T>> void searchTailRecursive( AVL<T> t, T x, Pair<T> p )

    {

        if( t == null ) return;

        if( AVL.rootValue(t).compareTo(x) <= 0 )

        {

            p.le = t;

            searchTailRecursive(AVL.right(t), x, p);

        }

        else

        {

            p.gt = t;

            searchTailRecursive(AVL.left(t),x,p);

        }

    }

    // Notkun: searchLoop(t,x,p);

    // Fyrir:  t er AVL<T> tré, x er T gildi, p er Pair<T>.

    //         Ekkert þessara gilda er null og t er í vaxandi

    //         röð.

    // Eftir:  p.le vísar á aftasta hnút í t með gildi <=x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.le jafnt null.

    //         p.gt vísar á fremsta hnút í t með gildi >x,

    //         ef slíkur hnútur er til. Ef slíkur hnútur er

    //         ekki til þá er p.gt jafnt null.

    public static<T extends Comparable<? super T>> void searchLoop( AVL<T> t, T x, Pair<T> p )

    {

        AVL<T> s = t;

        while( s != null )

        {

            // s er undirtré t með einhverja trjáslóð sp.

            // Allt í PreSeq(t,sp) er <=x.

            // Allt í PostSeq(t,sp) er >x.

            // Ef til er hnútur í t fyrir framan s undirtréð

            // (þ.e. PreSeq(t,sp) er ekki tómt) þá vísar

            // p.le á aftasta slíkan hnút.  Annars er p.le

            // jafnt null.

            // Ef til er hnútur í t fyrir aftan s undirtréð

            // (þ.e. PostSeq(t,sp) er ekki tómt) þá vísar

            // p.gt á fremsta slíkan hnút.  Annars er p.gt

            // jafnt null.

            if(AVL.max(t).compareTo(x) < 0){

                p.gt = null;

            }

            if(AVL.min(t).compareTo(x) > 0) {

                p.le = null;

            }

            if( AVL.rootValue(s).compareTo(x) <= 0 )

            {

                p.le = s;

                s = AVL.right(s);

            }

            else {

                p.gt = s;

                s = AVL.right(s);

            }

        }

    }

    private static void test\_LastLE\_FirstGT

        ( java.util.function.Function<Integer,Pair<Integer>> f

        , AVL<Integer> t

        )

    {

        for( int x=-1 ; x!=2000 ; x++ )

        {

            Pair<Integer> p = f.apply(x);

            if( p.le != null )

            {

                if( AVL.rootValue(p.le) > x )

                    throw new Error(""+x);

                if( x >= 1998 && AVL.rootValue(p.le) != 1998 )

                    throw new Error(""+x);

                if( x <= 1998 && x/2\*2 != AVL.rootValue(p.le) )

                    throw new Error(""+x);

                if( AVL.right(p.le) != null && AVL.rootValue(AVL.right(p.le)) <= x )

                    throw new Error(""+x);

            }

            else

            {

                if( x >= 0 ) throw new Error(""+x);

                if( AVL.find(t,x) || AVL.find(t,x-1) ) throw new Error(""+x);

            }

            if( p.gt != null )

            {

                if( AVL.rootValue(p.gt) <= x )

                    throw new Error(""+x);

                if( x < 0 && AVL.rootValue(p.gt) != 0 )

                    throw new Error(""+x);

                if( x >= 0 && (x+2)/2\*2 != AVL.rootValue(p.gt) )

                    throw new Error(""+x);

                if( AVL.left(p.gt) != null && AVL.rootValue(AVL.left(p.gt)) > x )

                    throw new Error(""+x);

            }

            else

            {

                if( x < 1998 ) throw new Error(""+x);

                if( AVL.find(t,x+1) || AVL.find(t,x+2) ) throw new Error(""+x);

            }

        }

    }

    public static void main( String[] args )

    {

        AVL<Integer> t = null;

        for( int n=0 ; n!=10 ; n++ )

            for( int i=0 ; i!=1000 ; i++ )

                t = AVL.insert(t,2\*i);

        final AVL<Integer> s = t;

        final Pair<Integer> p = new Pair<Integer>();

       try

        {

            test\_LastLE\_FirstGT

                            ( i ->

                                {

                                    searchRecursive(s,i,p);

                                    return p;

                                }

                            , s

                            );

        }

        catch( Error e )

        {

            System.out.println("Villa í searchRecursive í leit að "+e.getMessage());

        }

        try

        {

            test\_LastLE\_FirstGT

                            ( i ->

                                {

                                    p.le = p.gt = null;

                                    searchTailRecursive(s,i,p);

                                    return p;

                                }

                            , s

                            );

        }

        catch( Error e )

        {

            System.out.println("Villa í searchTailRecursive í leit að "+e.getMessage());

        }

        try

        {

            test\_LastLE\_FirstGT

                            ( i ->

                                {

                                    searchLoop(s,i,p);

                                    return p;

                                }

                            , s

                            );

        }

        catch( Error e )

        {

            System.out.println("Villa í searchLoop í leit að "+e.getMessage());

        }

        System.out.println("Prófunum lokið");

    }

}

