

Greining reiknirita: Verkefni 1

Kennari: Páll Melsted

Bjarki Geir Benediktsson

Bjarni Jens Kristinsson

Tandri Gauksson

Skil: 9. mars 2014

1 Lýsing á gagnagrind

Verkefnið var að hanna gagnagrind sem byggir á einhverskonar tréi og geymir lokað bil á rauntalnalínunni þar sem báðir endanpunktarnir eru heiltölur. Notast var við AVL tré, en AVL tré eru tvíleitartré sem uppfylla að auki skilyrðið: *Fyrir sérhvern hnút er mismunur á hæðum hægra og vinstra hluttrés í mesta lagi einn*. Því skilyrði má viðhalda með því að framkvæma snúninga á tréinu í hvert sinn sem innsetning eða eyðing valda ójafnvægi.

Fyrir hnút N látum við T_N tákna tréð sem hefur N sem rót. Í N eru eftirfarandi upplýsingar geymdar:

- Lokað bil, $N.I$,
- $N.Min$ = Lægsti endapunktur meðal þeirra bila sem eru innihaldin í T_N ,
- $N.Max$ = Hæsti endapunktur meðal þeirra bila sem eru innihaldin í T_N ,
- $N.h$ = Hæð T_N ,
- $N.l$ = Bendir á vinstra barn (ef það er til),
- $N.r$ = Bendir á hægra barn (ef það er til).

Við segjum að tvö bil skerist ef sniðmengi þeirra er ekki tómt. Við segjum að bil J skeri tréð T_N ef bilin J og $[N.Min, N.Max]$ skerast (ekkert bil sker tómt tré). Til þess að finna öll bil í T_N sem skera bilið J var búið til nýtt tré S og eftirfarandi endurkvæma reiknirit notað:

1. Ef $N.I$ og J skerast er bilinu $N.I$ fært inn í S .
2. Ef J sker $T_{N.l}$ er reikniritið notað á $T_{N.l}$.
3. Ef J sker $T_{N.r}$ er reikniritið notað á $T_{N.r}$.

Þegar keyrslu lýkur inniheldur S nákvæmlega þau bil í T_N sem skera J .

Þetta sama reiknirit má svo nota til þess að útfæra aðrar aðferðir. Til þess að finna öll bil í T_N sem innihalda tölu k má í staðinn leita að öllum bilum í T_N sem skera bilið $[k, k]$. Til þess að finna öll bil í T_N sem skera bilið $[a, b]$ má finna tréð S af öllum bilum í T_N sem innihalda töluna a og finna svo öll bil í S sem innihalda töluna b .

2 Java kóði

2.1 AVLIntervalTree

```
class AVLIntervalTree {  
    private Interval Value;
```

```
private AVLIntervalTree left;
private AVLIntervalTree right;
private int height;
private int Max;
private int Min;

/*
 * Fastayrðing gagna:
 *
 * Skilgreinum null.height := 0 og null.Max := -infinity.
 *
 * Ef left = null er Min = Value.min()
 * Annars er Min = left.Min()
 *
 * Max = max{ left.Max, Value.max(), right.Max }
 *
 * height = max{ left.height, right.height } + 1
 *
 * Tvíleitarskilyrði:
 * Ef left != null er left.Value < Value
 * Ef right != null er Value < right.Value
 *
 * AVL-skilyrði:
 * |left.height - right.height| <= 1
 *
 */

// Notkun: m = maxof3( a, b, c )
// Fyrir:
// Eftir: m = max{ a, b, c }
private static int maxof3( int a, int b, int c ) {
    int max = a;
    if( max < b ) max = b;
    if( max < c ) max = c;
    return max;
}

// Notkun: m = minof2( a, b );
// Fyrir:
// Eftir: m = min{ a, b }
private static int minof2( int a, int b ) {
    int min = a;
    if( min > b ) min = b;
    return min;
}

// Notkun: updateMax( T );
// Fyrir: T != null
// Eftir: T.Max = max{ left.Max, T.Value.max(), right.Max }
private static void updateMax( AVLIntervalTree T ) {
    int a, b;
    a = Integer.MIN_VALUE;
    b = a;
    if( T.left != null ) a = T.left.Max;
```

```
        if( T.right != null ) b = T.right.Max;
        T.Max = maxof3( a, b, T.Value.max() );
    }

    // Notkun: updateMin( T );
    // Fyrir: T != null
    // Eftir: T.Min = min{ left.Min, T.Value.min() }, þar sem null.Min = infinity
    private static void updateMin( AVLIntervalTree T ) {
        int a, b;
        a = Integer.MAX_VALUE;
        if( T.left != null ) a = T.left.Min;
        T.Min = minof2( a, T.Value.min() );
    }

    // Notkun: S = containInterval( T, I );
    // Fyrir:
    // Eftir: S inniheldur öll bil úr T sem innihalda bilið I.
    public static AVLIntervalTree containInterval( AVLIntervalTree T, Interval I ) {
        if( T == null ) return null;
        AVLIntervalTree R = null, S = null;
        S = containInteger( T, I.min() );
        R = containInteger( S, I.max() );
        return R;
    }

    // Notkun: S = containInteger( T, k )
    // Fyrir:
    // Eftir: S inniheldur öll bil úr T sem innihalda k.
    public static AVLIntervalTree containInteger( AVLIntervalTree T, int k ) {
        if( T == null ) return null;
        Interval I = new Interval(k,k);
        return intersectInterval(T,I);
    }

    // Notkun: S = intersectInterval( T, I )
    // Fyrir:
    // Eftir: S inniheldur öll bil úr T sem skera bilið I.
    public static AVLIntervalTree intersectInterval( AVLIntervalTree T, Interval I ) {
        if( T == null ) return null;
        AVLIntervalTree S = null;
        S = findIntersecting(S, T, I);
        return S;
    }

    // Notkun: S2 = findIntersecting( S, T, I )
    // Fyrir: T != null
    // Eftir: S2 inniheldur sömu bil og S og að auki öll bil úr T sem skera bilið I.
    private static AVLIntervalTree findIntersecting( AVLIntervalTree S, AVLIntervalTree T, Interval I ) {
        if( T.Value.intersects(I) )
            S = insert(S, T.Value);
        if( T.left != null && T.left.Max >= I.min() && T.left.Min <= I.max() )
            S = findIntersecting(S, T.left, I);
        if( T.right != null && T.right.Max >= I.min() && T.right.Min <= I.max() )
            S = findIntersecting(S, T.right, I);
    }
```

```

    return S;
}

// Notkun: tree2 = rotateLL( tree );
// Fyrir: tree != null og tree.left != null
// Eftir: tree2 inniheldur sömu upplýsingar og tree, en búið er að snúa tree
//        og uppfæra height, Max og Min í tree2 og tree2.right skv. mynd.
//
//      x          y
//     /\         /\
//    y  C      => A  x
//   /\       /\
//  A  B      B  C
//
private static AVLIntervalTree rotateLL( AVLIntervalTree tree ) {
    AVLIntervalTree x = tree, y = x.left;
    x.left = y.right;
    x.height = height(x.left,x.right);
    updateMax(x);
    updateMin(x);
    y.right = x;
    y.height = height(y.left,x);
    updateMax(y);
    updateMin(y);
    return y;
}

// Notkun: tree2 = rotateLR( tree );
// Fyrir: tree != null, tree.left != null og tree.left.right != null
// Eftir: tree2 inniheldur sömu upplýsingar og tree, en búið er að snúa tree
//        og uppfæra height, Max og Min í tree2, tree2.left og tree2.right skv. mynd.
//
//      x          z
//     /\         /\
//    y  D      => y  x
//   /\       /\  /\
//  A  z      A  B C D
//   /\
//  B  C
//
private static AVLIntervalTree rotateLR( AVLIntervalTree tree ) {
    AVLIntervalTree x = tree, y = x.left, z = y.right;
    x.left = z.right;
    x.height = height(x.left,x.right);
    updateMax(x);
    updateMin(x);
    y.right = z.left;
    y.height = height(y.left,y.right);
    updateMax(y);
    updateMin(y);
    z.right = x;
    z.left = y;
    z.height = height(y,x);
    updateMax(z);
    updateMin(z);
    return z;
}

```

```

// Notkun: tree2 = rotateLL( tree );
// Fyrir: tree != null og tree.right != null
// Eftir: tree2 inniheldur sömu upplýsingar og tree, en búið er að snúa tree
//        og uppfæra height, Max og Min í tree2 og tree2.left skv. mynd.
//
//      x          y
//     /\         /\
//    A  y      => x  C
//     /\         /\
//    B  C      A  B
private static AVLIntervalTree rotateRR( AVLIntervalTree tree ) {
    AVLIntervalTree x = tree, y = x.right;
    x.right = y.left;
    x.height = height(x.left,x.right);
    updateMax(x);
    updateMin(x);
    y.left = x;
    y.height = height(x,y.right);
    updateMax(y);
    updateMin(y);
    return y;
}

// Notkun: tree = rotateRL( tree );
// Fyrir: tree != null, tree.right != null og tree.right.left != null
// Eftir: tree2 inniheldur sömu upplýsingar og tree, en búið er að snúa tree
//        og uppfæra height, Max og Min í tree2, tree2.left og tree2.right skv. mynd.
//
//      x          z
//     /\         /\
//    A  y      => x  y
//     /\         /\  /\
//    z  D      A  B C D
//     /\
//    B  C
private static AVLIntervalTree rotateRL( AVLIntervalTree tree ) {
    AVLIntervalTree x = tree, y = x.right, z = y.left;
    x.right = z.left;
    x.height = height(x.left,x.right);
    updateMax(x);
    updateMin(x);
    y.left = z.right;
    y.height = height(y.left,y.right);
    updateMax(y);
    updateMin(y);
    z.left = x;
    z.right = y;
    z.height = height(x,y);
    updateMax(z);
    updateMin(z);
    return z;
}

// Notkun: tree = AVLIntervalTree(I);
// Fyrir:

```

```
// Eftir: tree er AVLIntervalTree hlutur með tree.Value = I en enga aðra hnúta.
private AVLIntervalTree( Interval I ) {
    Value = I;
    height = 1;
    Max = I.max();
    Min = I.min();
}

// Notkun: h = height(tree);
// Fyrir:
// Eftir: h = tree.height
static int height( AVLIntervalTree tree ) {
    if( tree==null ) return 0;
    return tree.height;
}

// Notkun: h = height(left,right);
// Fyrir:
// Eftir: h = max{ left.height, right.height } + 1
static int height( AVLIntervalTree left, AVLIntervalTree right ) {
    int leftheight = height(left);
    int rightheight = height(right);
    if( leftheight > rightheight )
        return leftheight+1;
    else
        return rightheight+1;
}

// Notkun: f = find(tree,I);
// Fyrir:
// Eftir: f == true þþaa I er í tree
public static boolean find( AVLIntervalTree tree, Interval I ) {
    if( tree==null )
        return false;
    if( tree.Value.equals(I) )
        return true;
    if( I.compareTo(tree.Value) < 0 )
        return find(tree.left,I);
    else
        return find(tree.right,I);
}

// Notkun: I = min(tree);
// Fyrir:
// Eftir: I er fremsti Interval hluturinn í tree.
static Interval min( AVLIntervalTree tree ) {
    if( tree==null ) return null;
    if( tree.left==null ) return tree.Value;
    return min(tree.left);
}

// Notkun: I = max(tree);
// Fyrir:
// Eftir: I er aftasti Interval hluturinn í tree.
```

```
static Interval max( AVLIntervalTree tree ) {
    if( tree==null ) return null;
    if( tree.right==null ) return tree.Value;
    return max(tree.right);
}

// Notkun: tree = insert(org,I);
// Fyrir:
// Eftir: tree inniheldur allar sömu upplýsingar og org,
//        en auk þess hefur einn hnútur gildið I.
public static AVLIntervalTree insert( AVLIntervalTree org, Interval I ) {
    if( org==null )
        return new AVLIntervalTree(I);

    // Uppfæri org.Max og org.Min
    if( I.max() > org.Max ) org.Max = I.max();
    if( I.min() < org.Min ) org.Min = I.min();

    if( I.compareTo(org.Value) < 0 ) {
        org.left = insert(org.left,I);

        // org.left != null
        if(height(org.left) > height(org.right) + 1) {
            if(I.compareTo(org.left.Value) >= 0) // org.left.right != null
                org = rotateLR(org);
            else org = rotateLL(org);
        }
        org.height = height(org.left,org.right);
    }
    else if( I.compareTo(org.Value) > 0 ) {
        org.right = insert(org.right,I);

        // org.right != null
        if(height(org.right) > height(org.left) + 1) {
            if(I.compareTo(org.right.Value) < 0) // org.right.left != null
                org = rotateRL(org);
            else org = rotateRR(org);
        }
        org.height = height(org.left,org.right);
    }
    return org;
}

// Notkun: tree = delete(org,I);
// Fyrir:
// Eftir: tree inniheldur allar sömu upplýsingar og org,
//        nema enginn hnútur hefur gildið I.
public static AVLIntervalTree delete( AVLIntervalTree org, Interval I ) {
    if( org==null ) return null;
    if( I.equals(org.Value) ) {
        if( height(org.left) > height(org.right) ) {
            org.Value = max(org.left);
            org.left = delete(org.left,org.Value);
        }
    }
}
```

```
// Uppfæri org.Max
updateMax(org);

if( height(org.left) + 1 < height(org.right) ) {
    if( height(org.right.right) < height(org.right.left) ) org = rotateRL(org);
    else org = rotateRR(org);
}

org.height = height(org.left,org.right);
return org;
}
else if( org.right != null ) {
    org.Value = min(org.right);
    org.right = delete(org.right,org.Value);

    // Uppfæri org.Max og org.Min
    updateMax(org);
    updateMin(org);

    if( height(org.right) + 1 < height(org.left) ) {
        if( height(org.left.left) < height(org.left.right) ) org = rotateLR(org);
        else org = rotateLL(org);
    }

    org.height = height(org.left,org.right);

    return org;
}
else
    return null;
}
if( I.compareTo(org.Value) < 0 ) {
    org.left = delete(org.left,I);
    // Uppfæri org.Max og org.Min
    updateMax(org);
    updateMin(org);

    if( height(org.left) + 1 < height(org.right) ) {
        if( height(org.right.right) < height(org.right.left) ) org = rotateRL(org);
        else org = rotateRR(org);
    }

    org.height = height(org.left,org.right);
}
else {
    org.right = delete(org.right,I);
    // Uppfæri org.Max
    updateMax(org);

    if( height(org.right) + 1 < height(org.left) ) {
        if( height(org.left.left) < height(org.left.right) ) org = rotateLR(org);
        else org = rotateLL(org);
    }
}
```



```
        org.height = height(org.left,org.right);
    }
    return org;
}

// Notkun: b = checkAVL( tree );
// Fyrir:
// Eftir: b == true þþaa tree uppfylli AVL skilyrði.
public static boolean checkAVL( AVLIntervalTree tree ) {
    if( tree == null ) return true;
    int d = height(tree.left) - height(tree.right);
    return d > -2 && d < 2 && checkAVL(tree.left) && checkAVL(tree.right);
}

// Notkun: b = checkBST( tree );
// Fyrir:
// Eftir: b = true þþaa tree uppfylli Tvíleitaraskilyrði.
public static boolean checkBST( AVLIntervalTree tree ) {
    if( tree == null ) return true;
    boolean c = tree.left == null || tree.left.Value.compareTo(tree.Value) <= 0;
    boolean d = tree.right == null || tree.right.Value.compareTo(tree.Value) >= 0;
    return c && d && checkBST( tree.left ) && checkBST( tree.right );
}

// Notkun: s = T.toString()
// Fyrir:
// Eftir: s er strengur af bilum T í vaxandi röð.
public String toString() {
    StringBuilder sb = new StringBuilder();
    buildStringFromTree(this,sb);
    return sb.toString();
}

// Notkun: buildStringFromTree(T,sb);
// Fyrir: T != null
// Eftir: Búið er að bæta við öllum strengjum í trénu T inn í sb í stafrófsröð.
private static void buildStringFromTree(AVLIntervalTree T, StringBuilder sb) {
    if (T.left != null) buildStringFromTree(T.left,sb);
    sb.append(T.Value.toString()+" ");
    if (T.right!= null) buildStringFromTree(T.right,sb);
}
}
```

2.2 Interval

```
public class Interval implements Comparable<Interval> {
    private final int left;
    private final int right;
    // Fastayrðing gagna: engin

    // Notkun: I = new Interval(a,b)
```

```
// Fyrir: Ekkert
// Eftir: I = [a, b]
public Interval(int a, int b){
    if( a <= b ){
        left = a;
        right = b;
    }
    else{
        left = 1;
        right = -1;
    }
}

// Notkun: tomt = I.isEmpty()
// Fyrir: Ekkert
// Eftir: tomt = true ef I er tómamengið, tomt = false annars
public boolean isEmpty(){
    return right < left;
}

// Notkun: max = I.max();
// Fyrir: Ekkert
// Eftir: max eru efri mörk I
public int max(){
    return right;
}

// Notkun: min = I.min();
// Fyrir: Ekkert
// Eftir: min eru neðri mörk I
public int min(){
    return left;
}

// Notkun: inni = I.contains(x)
// Fyrir: Ekkert
// Eftir: inni = true ef x er stak í I, inni = false annars
public boolean contains(int x){
    return (left <= x) && (x <= right);
}

// Notkun: inni = I.contains(J)
// Fyrir: Ekkert
// Eftir: inni = true ef J er allt innihaldið í I, inni = false annars
public boolean contains(Interval J){
    return J.isEmpty() || (left <= J.min()) && (J.max() <= right);
}

// Notkun: sker = I.intersects(J)
// Fyrir: Ekkert
// Eftir: sker = true ef sniðmengi I og J er ekki tomt,
//        sker = false annars
public boolean intersects(Interval J){
    if((right < left) || J.isEmpty()) return false;
}
```

```
        return ((right >= J.left) || (right >= J.right)) && ((J.left >= left) || (J.right >= left))
    }

    // Notkun: s = I.toString()
    // Fyrir: Ekkert
    // Eftir: s er strengur á forminu "[left, right]"
    public String toString(){
        return "[" + left + ", " + right + "]";
    }

    // Notkun: c = I.compareTo(J)
    // Fyrir: Ekkert
    // Eftir: c > 0 ef I < J
    //         c < 0 ef I > J
    //         c = 0 ef I = J.
    public int compareTo(Interval J){
        if( left < J.min() || (left == J.min() && right < J.max()) ){
            return -1;
        }
        if( left > J.min() || (left == J.min() && right > J.max()) ){
            return 1;
        }
        return 0;
    }

    // Notkun: b = I.equals(J)
    // Fyrir: Ekkert
    // Eftir: b = true þáa I = J
    public boolean equals(Interval J){
        return left == J.min() && right == J.max();
    }
}
```

2.3 PrufuForrit

```
//~ Við keyrum profurorritið með skipuninni
//~ time java PrufuForrit < io/s1.in | diff -w io/s1.out - | wc -l

public class PrufuForrit {

    // Notkun: s = treeToString(T);
    // Fyrir: (ekkert)
    // Eftir: s er strengurinn "[]" ef T er null, annars er s strengurinn T.toString().
    private static String treeToString( AVLIntervalTree T ) {
        if (T==null) return "[]";
        else return T.toString();
    }

    public static void main(String[] args) {
        //~ Skanni sem les inn tölurnar
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        //~ tree==null sem þýðir að tree vísar í tómtré.
        AVLIntervalTree tree = null;
    }
}
```

```
//~ F: Búið er að lesa inn núll línur af staðalinntaki.
while (scanner.hasNext()) {
    //~ I: Búið er að lesa inn núll eða fleiri línur af staðalinntaki og gera viðeigandi
    String line = scanner.nextLine(); //tekur inntak fram að næsta enter
    String[] S = line.split(" "); //býr til fylki af orðunum í línunni (skipt eftir bilum)

    //~ Skoðum núna hvert fyrsta orðið í línunni er.
    switch (S[0]) {
        case "+": assert(S.length==3); // "+ a b"
                    tree = AVLIntervalTree.insert( tree, new Interval( Integer.parseInt(S[1]), Integer.parseInt(S[2])) );
                    break;
        case "-": assert(S.length==3); // "- a b"
                    tree = AVLIntervalTree.delete( tree, new Interval( Integer.parseInt(S[1]), Integer.parseInt(S[2])) );
                    break;
        case "?o": assert(S.length==3); // "?o a b"
                    System.out.println( treeToString( AVLIntervalTree.intersectInterval( tree, new Interval( Integer.parseInt(S[1]), Integer.parseInt(S[2])) ) ) );
                    break;
        case "?i": assert(S.length==3); // "?i a b"
                    System.out.println( treeToString( AVLIntervalTree.containInterval( tree, new Interval( Integer.parseInt(S[1]), Integer.parseInt(S[2])) ) ) );
                    break;
        case "?p": assert(S.length==2); // "?p a"
                    System.out.println( treeToString( AVLIntervalTree.containInteger( tree, Integer.parseInt(S[1]) ) ) );
                    break;
        default: System.out.println("Input line in wrong format. Exiting now."); System.exit(1);
    }
}
//~ E: Búið er að lesa inn allar línur af staðalinntaki og gera viðeigandi aðgerðir með þeim
}
```