Greining Reiknirita

TÖL403G

Skilaverkefni 2

Verkefnishöfundar:
Guðmundur Már Gunnarsson
Skarphéðinn Þórðarsson
Sigurður Skúli Sigurgeirsson

Kennari:
Páll Melsted

7. mars 2014

```
port java.util.Scanner;

blic class IntervalTree {

/nóðurnar í trénu
tatic class Node

int q; //miðgildið í bilinu
int lower; //neðri mörk
int higher; //efri mörk
Node left; //vinstra barn
Node right; //hægra barn
Node parent; //foreldri
Link intervals; //bilin sem skerast á við bilið í nóðunni

//Notkun: node.insertInterval(a,b);
//Fyrir: a og b eru heiltölur, a < b
//Eftir: búið er að setja bilið [a,b] á réttan stað í intervals
void insertInterval(int a, int b)
{
        Link newLink = new Link();
        newLink.lower = a;
        newLink.higher = b;
        if( intervals == null || intervals.compareTo(a,b)<0 )
        {
          newLink.next=intervals;
          intervals=newLink;
          return;
        }
        if(intervals.lower == a && intervals.higher == b)
        {
                return;
        }
        Link temp = intervals;
        while( temp.next != null )
        {
                if( temp.next.compareTo(a,b) > 0 )
                {
                        temp = temp.next;
                }
                else if( temp.next.compareTo(a,b) == 0 )
                        return;
                else
                {
                        newLink.next = temp.next;
                        temp.next = newLink;
                        return;
                }
        }
        newLink.next = temp.next;
        temp.next = newLink;
}
```

```

//Notkun: node.findIntersections(a,b);
//Fyrir: a og b eru heiltölur, a < b
//Eftir: búið er að finna öll bil sem skerast á við bilið [a,b]
int findIntersections(int a,int b)
{
        Link chain = intervals;

        int found = 0;

        while(chain != null)
        {
                if((chain.lower <= b && chain.higher >= b) || (chain.lower <= a && chain.higher >= a))
                {
                        System.out.print("["+chain.lower+", "+chain.higher+"] ");
                        found++;
                }
                else if((chain.lower >= a && chain.lower <= b) || (chain.higher >= a && chain.higher <=
                {
                        System.out.print("["+chain.lower+", "+chain.higher+"] ");
                        found++;
                }
                chain = chain.next;
        }

        return found;
}

//Notkun: node.findContains(a,b);
//Fyrir: a og b eru heiltölur, a < b
//Eftir: búið er að finna öll bil sem innihalda [a,b]
boolean findContains(int a,int b)
{
        Link chain = intervals;

        boolean found = false;

        while(chain != null)
        {
                if(chain.lower <= a && b <= chain.higher)
                {
                        System.out.print("["+chain.lower+", "+chain.higher+"] ");
                        found = true;
                }
                chain = chain.next;
        }

        return found;
}

void deleteInterval(int a, int b)
{
        if(intervals == null) return;

        //athugar hvort fremsta stakið sé það sem verið er að leita af
        if(intervals.lower == a && intervals.higher == b)
        {
                intervals = intervals.next;
                return;
        }

        Link chain = intervals;

        //fer í gegnum afganginn af listanum og leitar
        while(chain.next != null)
        {
                if(chain.next.lower == a && chain.next.higher == b)
                {
                        chain.next = chain.next.next;
                        return;
                }

                chain = chain.next;
        }
}



tatic class Link {
Link next;
int lower;
int higher;

//Notkun: link.compareTo(a,b);
//Fyrir: a og b eru heiltölur, a < b
//Eftir: Skilar 1 ef [lower,higher] < [a,b], 0 ef þau eru jöfn og -1 annars
int compareTo(int a, int b)
{
        if(lower < a)
        {
                return 1;
        }
        else if(lower > a)
        {
                return -1;
        }
        else
        {
                if(higher < b)
                {
                        return 1;
                }
                else if(higher > b)
                {
                        return -1;
                }
```

```
                                else
                                {
                                        return 0;
                                }
                        }
                }


ode root;

ublic IntervalTree()

root = null;


/Notkun: tree.insert(a,b);
/Fyrir: a og b eru heiltölur, a < b
/Eftir: búið er að bæta bilinu [a,b] í tréð
ublic void insert(int a, int b)

if(b < a) return;

Node newNode = new Node();
newNode.q = a+b/2;
newNode.lower = a;
newNode.higher = b;
newNode.insertInterval(a,b);

if(root == null) {
        root = newNode;
        return;
}

Node tree = root;

while(tree != null)
{
        if(b < tree.lower)
        {
                if(tree.left != null)
                {
                        tree = tree.left;
                }
                else
                {
                        newNode.parent = tree;
                        tree.left = newNode;
                        return;
                }
        }
        else if(a > tree.higher)
        {
                if(tree.right != null)
                {
                        tree = tree.right;
                }
                else
                {
                        newNode.parent = tree;
                        tree.right = newNode;
                        return;
                }
        }
        else
        {
                tree.insertInterval(a,b);
                return;
        }
}


/Notkun: tree.intersects(a,b,root);
/Fyrir: a og b eru heiltölur, a < b, root er nóða
/Eftir: búið er að finna öll bil sem skerast á við [a,b]
ublic int intersects(int a, int b, Node node)

if(b < a) return 0;


if(node == null)
{
        return 0;
}

int instanceFound = 0;

Node tree = node;

if(a < tree.lower)
{
        instanceFound = instanceFound + intersects(a,b, tree.left);
}

instanceFound = instanceFound + node.findIntersections(a,b);

if(b > tree.higher)
{
        instanceFound = instanceFound + intersects(a,b, tree.right);
}
return instanceFound;


ublic void intersects(int a, int b)

int instance = intersects(a, b, root);
if(instance == 0)
```

```
{
        System.out.print("[]");
}
System.out.println("");


/Notkun: tree.contains(a,b,root);
/Fyrir: a og b eru heiltölur, a <= b, root er nóða
/Eftir: búið er að finna öll bil sem innihalda[a,b]
ublic boolean contains(int a, int b, Node node)

if(b < a) return false;

if(node == null)
{
        return false;
}

boolean instanceFound = false;

Node tree = node;

if(a < tree.lower)
{
        boolean left =  contains(a,b, tree.left);
        instanceFound = instanceFound || left;
}

boolean center = tree.findContains(a,b);
instanceFound = instanceFound || center;

if(b > tree.higher)
{
        boolean right = contains(a,b, tree.right);
        instanceFound = instanceFound || right;
}

return instanceFound;


ublic void contains(int a, int b)

boolean instance = contains(a, b, root);
if(!instance)
{
        System.out.print("[]");
}
System.out.println("");


/Notkun: tree.point(a);
/Fyrir: a er heiltala
/Eftir: búið er að finna öll bil sem innihalda a
```

```
ublic void point(int a)

boolean instance = contains(a, a, root);
if(!instance)
{
        System.out.print("[]");
}
System.out.println("");


/Notkun: tree.delete(a,b);
/Fyrir: a og b eru heiltölur, a <= b
/Eftir: Ef [a,b] var í trénu þá er búið að eyða því
ublic void delete(int a, int b)


if(root == null || b < a) return;

Node tree = root;

while(tree != null)
{
        if(b < tree.lower)
        {
                if(tree.left != null)
                {
                        tree = tree.left;
                }
                else
                {
                        return;
                }
        }
        else if(a > tree.higher)
        {
                if(tree.right != null)
                {
                        tree = tree.right;
                }
                else
                {
                        return;
                }
        }
        else
        {
                tree.deleteInterval(a,b);
                if(tree.intervals == null)
                {
                        deleteNode(tree);
                }
                return;
        }
}


/Notkun: tree.deleteNode(node)
/Fyrir: node er nóða
/Eftir: Búið er að fjarlægja node úr trénu
ublic void deleteNode(Node node)



if(node == null) return;

if(node.left == null && node.right == null)
{
        node = null;
        return;
}
if(node.right == null)
{
        node.left.parent = node.parent;
        node = node.left;
        return;
}

Node search = node.right;
while(search.left != null)
{
        search = search.left;
}

Node copyOfSearch = search;

search = search.right;
copyOfSearch.parent = node.parent;
node = copyOfSearch;




/gengur í gegnum tréð, bara aðstoðarfall ekki skila
ublic void traverse(Node root)

if(root == null)
{
        return;
}
System.out.println(root.lower+" : "+root.higher);
traverse(root.left);
traverse(root.right);


ublic static void main(String[] args)

```

```java
IntervalTree tree = new IntervalTree();
Scanner scanner = new Scanner(System.in);
while(scanner.hasNext())
{
        String query = scanner.nextLine();
        String[] splitQuery = query.split(" ");
        int lower = Integer.parseInt(splitQuery[1]);

        if(splitQuery[0].equals("?p"))
        {
                tree.point(lower);
        }
        else {

                int higher = Integer.parseInt(splitQuery[2]);

                if(splitQuery[0].contains("+"))
                {
                        tree.insert(lower, higher);
                }
                else if(splitQuery[0].equals("?o"))
                {
                        tree.intersects(lower,higher);
                }
                else if(splitQuery[0].equals("?i"))
                {
                        tree.contains(lower,higher);
                }
                else if(splitQuery[0].equals("-"))
                {
                        tree.delete(lower, higher);
                }

        }


}
```