University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

February 25, 2020

Homework 7 problem 4

Programming Problem 4 (10 points) Greedy Algorithms for Interval Scheduling:

This programming problem and the next will look at the interval scheduling problem with the objective function of maximizing the sum of the lengths of selected intervals: The input for an interval scheduling problem is a set of intervals $I = \{i_1, \ldots, i_n\}$ where i_k has start time s_k, and finish time f_k and the output is a set of non-overlapping intervals that has the maximum possible sum of lengths.

Implement routines for the following:

a) A random interval generator. Given integer parameters n, L, and r, generate n intervals, where each interval has a starting position uniformly chosen from $[1, L]$ and length uniformly chosen from $[1, r]$.

b) A greedy algorithm for interval scheduling which selects intervals in earliest starting time first order.

c) A greedy algorithm for interval scheduling which selects intervals in longest length first order.

For this problem, submit your code for the three routines.

Answer:

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Random;

public class Interval {
    public int index;
    public int startTime;
    public int endTime;
    public int length;
    public boolean exluded = false;

    public Interval(int s, int l){
        this.startTime = s;
        this.length = l;
        this.endTime = s + l;
    }
```

```
    @Override
    public String toString(){
        return String.format("%d:%d–%d(%d)",
        index, startTime, endTime, length);
    }

}

class IntervalSet{
    public int n;
    public Interval[] intervals;

    // part a
    public IntervalSet(int n, int L, int r) {
        this.intervals = new Interval[n];
        this.n = n;
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            int s = rand.nextInt(L) + 1;
            int l = rand.nextInt(r) + 1;
            this.intervals[i] = new Interval(s, l);
        }
        Arrays.sort(this.intervals, new startTimeComparator());
        for (int i = 0; i < n; i++) {
            this.intervals[i].index = i;
        }
    }

    public int nextCompatiInv(Interval inv){
        int i = Arrays.binarySearch(this.intervals, inv,
        (Interval i1, Interval i2) -> i1.startTime - i2.endTime);

        if (i < 0) {
            return -i-1;
        }
        return i;
    }

    public int lastCompatInv(Interval inv){
        int i = Arrays.binarySearch(this.intervals, inv,
        (Interval i1, Interval i2) -> i1.endTime - i2.startTime);

        if (i >= 0){
            return i;
```

```
            }
            // if no leftmost compatible interval found, return -1
            return Math.max(-i-2, -1);
        }

        // part b
        public LinkedList<Interval> StartTimeFirstSolution(){
        LinkedList<Interval> subset = new LinkedList<>();
        for (int i = 0; i < this.n;){
            Interval curInv = this.intervals[i];
            subset.add(curInv);
            i = this.nextCompatiInv(curInv);
        }
        return subset;
        }

        // part c
        public LinkedList<Interval> LongestLengthFirstSolution(){
        LinkedList<Interval> subset = new LinkedList<>();
        Interval[] intervalsByLen = this.intervals.clone();
        Arrays.sort(intervalsByLen, new lengthComparator());
        for (int i = 0; i < this.n; i++){
            Interval curInv = intervalsByLen[i];
            if (!curInv.exluded){
                subset.add(curInv);
                for (int j = curInv.index; j<this.nextCompatiInv(curInv); j ++){
                    this.intervals[j].exluded = true;
                }
                for (int j = this.lastCompatInv(curInv)+1; j<curInv.index; j ++){
                    this.intervals[j].exluded = true;
                }
            }
        }
        return subset;
        }
    }



class startTimeComparator implements Comparator<Interval> {
    @Override
    public int compare(Interval i1, Interval i2) {
        return i1.startTime - i2.startTime;
    }
}
```

```
class endTimeComparator implements Comparator<Interval> {
    @Override
    public int compare(Interval i1, Interval i2) {
        return i1.endTime - i2.endTime;
    }
}

class lengthComparator implements Comparator<Interval> {
    @Override
    public int compare(Interval i1, Interval i2) {
        return i2.length - i1.length;
    }
}
```