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

February 25, 2020

Homework 7 problem 5

Programming Problem 5 (10 points) Dynamic Programming for Interval Scheduling: Implement a dynamic programming algorithm that optimally solves the Interval Scheduling problem to maximize the sum of the lengths of non-overlapping intervals.

Evaluate the performance of the dynamic programming algorithm compared with the two greedy algorithms from Problem 4 on randomly generated intervals. In your test generator use $n = 10,000$, $L = 1,000,000$ and $r = 2,000$.

For this problem, submit your code for the dynamic programming problem along with the output from a series of test runs on all three algorithms.

Answer:

```
package problem5;

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);
    }

}
```

```java
class IntervalSet{
    public int n;
    public Interval[] intervals;
    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);
    }

    public int StartTimeFirstSolution(){
        int maxLen = 0;
        for (int i = 0; i < this.n;){
            Interval curInv = this.intervals[i];
            maxLen += curInv.length;
            i = this.nextCompatiInv(curInv);
        }
        return maxLen;
    }

    public int LongestLengthFirstSolution(){
        int maxLen = 0;
        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){
                maxLen += curInv.length;
                // ignore the overlap intervals
                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 maxLen;
    }


    public int DynamicProgrammingSolution(){
        Arrays.sort(this.intervals, new endTimeComparator());
        int[] opts = new int[this.n + 1];
        for (int i = 1; i < this.n + 1; i++) {
            Interval curInv = this.intervals[i-1];
            int p_cur = this.lastCompatInv(curInv);
            if(p_cur != -1){
                opts[i] = Math.max(curInv.length + opts[p_cur+1], opts[i-1]);
            }else{
                opts[i] = Math.max(curInv.length, opts[i-1]);
            }
        }
        return opts[this.n];
    }


    public static void main(String[] args) {
    int n = 20;
    int sumSTF = 0;
    int sumLLF = 0;
    int sumDP = 0;

    for(int i = 0; i < n; i++){
        System.out.printf("case␣%d\n\n",i);
        IntervalSet set = new IntervalSet(10000, 1000000, 2000);
        int STF = set.StartTimeFirstSolution();
        int LLF = set.LongestLengthFirstSolution();
        int DP = set.DynamicProgrammingSolution();

        System.out.printf("Start␣Time␣First:␣␣␣␣␣%d\n", STF);
        System.out.printf("Longest␣Length␣First:␣%d\n", LLF);
        System.out.printf("Dynamic␣Programming:␣␣%d\n\n", DP);

        sumSTF += STF;
        sumLLF += LLF;
        sumDP += DP;
    }

    System.out.printf("Average␣Start␣Time␣First:␣␣␣␣␣%.2f\n",
                        sumSTF/20.0);
    System.out.printf("Average␣Longest␣Length␣First:␣%.2f\n",
                        sumLLF/20.0);
    System.out.printf("Average␣Dynamic␣Programming:␣␣%.2f\n\n",
                        sumDP/20.0);
    }
}

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;
        }
    }
```

```
case 0
Start Time First: 910806
Longest Length First: 914360
Dynamic Programming: 953942
case 1
Start Time First: 914008
Longest Length First: 931116
Dynamic Programming: 954315
case 2
Start Time First: 908996
Longest Length First: 889896
Dynamic Programming: 953824
case 3
Start Time First: 905521
Longest Length First: 894780
Dynamic Programming: 952702
case 4
Start Time First: 913265
Longest Length First: 893705
Dynamic Programming: 956259
case 5
Start Time First: 911067
Longest Length First: 913026
Dynamic Programming: 953682
case 6
Start Time First: 913595
Longest Length First: 911104
Dynamic Programming: 953421
case 7
Start Time First: 910690
Longest Length First: 891846
Dynamic Programming: 952350
case 8
Start Time First: 912948
Longest Length First: 894753
Dynamic Programming: 955059
case 9
Start Time First: 911895
Longest Length First: 895558
Dynamic Programming: 953720
case 10
```

```
Start Time First: 912766
Longest Length First: 898262
Dynamic Programming: 953400
case 11
Start Time First: 912098
Longest Length First: 902858
Dynamic Programming: 953444
case 12
Start Time First: 914172
Longest Length First: 910233
Dynamic Programming: 953874
case 13
Start Time First: 917108
Longest Length First: 883794
Dynamic Programming: 954519
case 14
Start Time First: 914187
Longest Length First: 919044
Dynamic Programming: 954683
case 15
Start Time First: 911266
Longest Length First: 915287
Dynamic Programming: 954195
case 16
Start Time First: 910796
Longest Length First: 885701
Dynamic Programming: 954659
case 17
Start Time First: 905669
Longest Length First: 907627
Dynamic Programming: 952825
case 18
Start Time First: 911494
Longest Length First: 911085
Dynamic Programming: 955811
case 19
Start Time First: 903408
Longest Length First: 912640
Dynamic Programming: 952997

Average Start Time First: 911287.75
Average Longest Length First: 903833.75
Average Dynamic Programming: 953984.05
```