Interval Partitioning Report

Bergar Simonsen, Daniel Vasile, Aleksandra Korach, Tanase Comboeanu

September 7, 2015

Results

Our implementation produces the expected results on all input–output file pairs. Table 1 shows an example output from our algorithm.

Start	End	Label
1	20	0
2	4	1
6	8	1
10	12	1
14	16	1
		2

Table 1: Example output

The output from the algorithm is validated against a model output handed out with the exercise. In order to verify that our algorithm gives correct results, we run the algorithm on all inputs given with the exercise, and compare the output with the model output. We are mostly interested in checking that our algorithm generates the same amount of resources (labels, classrooms) as the model output. Table 2 shows that for all input files, the results are identical with the model output, and we can verify that our algorithm runs correctly.

Input file	Model output	Own output
ip-1.in	2	2
ip-2.in	2	2
ip-3.in	4	4
ip-rand-1k.in	518	518
ip-rand-10k.in	5024	5024
ip-rand-100k.in	50212	50212
ip-rand-1M.in	500523	500523

Table 2: Validation results

Implementation details

Our overall structure is very simple. We hold our intervals (Jobs) in an array and our resources are contained in Priority Queue.

Initially we sort the intervals by their start time, which happens in $O(n\ log(n))$, since we are using a standard sorting algorithm from the Java Arrays library, implemented as Dual-Pivot Quick Sort.

Our Priority Queue adds new elements in time $O(log(n))$, as stated in Section 2.5 of Kleinberg and Tardos, *Algorithms Design*, Addison–Wesley 2005. The priority is defined as the 'earliest-free resource'. Checking if the moment when the 'earliest-free resource' overlaps with a currently considered interval from the sorted list takes $O(1)$.

From the above and the properties of the problem it follows, that these checks are performed n times, once per interval, and adding a new element happens at most d times, where d is the depth of the problem. This amounts to a running time of $O(n\ log(d))$. We can then further conclude the overall time complexity of our algorithm, including initial sorting, is:

$$O(n\,log(n) + n\,log(d)) = O(n\,log(n)), \quad n >= d$$