Data Structures and Algorithms
CSL2020

Homework 6
Darshit Jain (B19EE024)

February 4, 2021

1 Problem

We have been given one single classroom and **n** professors are interested in using it. Each professor (lets say professor i) has duration of class t_i and a deadline of d_i for his class. The time starts from 0 and the finish time of the class is denoted by f_i for the professor i. We measure the annoyance of professor i by the following function:

$$annoyance_i = max(f_i - d_i, 0)$$

Our aim is to come up with an algorithm which minimizes the maximum of annoyance among all the professors.

2 Algorithm

Input:
n (number of classes) is given followed by duration and deadline for each class

Working:
Initialize an array of size n where each array element is a *struct element* with properties *duration, deadline, order*.
Perform quicksort on the array based on the deadlines.
Perform quicksort based on the duration for array elements having same deadline.

```
void quickSortDeadline(prof_class classes[ ], int start, int end) {
    perform quicksort based on deadline
}

void quickSortDuration(prof_class classes[ ], int start, int end) {
    perform quicksort based on duration
}

int count = 0;
int temp;
```

```
while(count<n){
   temp = count;
   while (classes[temp].deadline == classes[temp + 1].deadline){
      temp++;
   }
   if (temp != count){
       call quickSortDuration(classes, count, temp);
   }
   count = temp + 1;
}

int start = 0, finish = 0, annoyance = 0;

for elements in classes array:
   finish = start + element.duration;
   annoyance = annoyance + MAX(finish - element.deadline, 0);
   start = finish;
```

This way we will be able to minimize the maximum of annoyance among all the professors.

Output: Sequence of classes and sum of annoyance of all the professors

3 Cost of Algorithm

The overall cost of the algorithm is $O(n \log n)$ as quicksort will have a time complexity of $O(n \log n)$ for an array of size n.
If we consider **q** unique deadlines, then average occurrence of each deadline becomes n/q.
So for each partition, the time complexity becomes $O((n/q) \log{(n/q)})$ and hence for q such partitions the time complexity becomes $O(n \log{(n/q)})$.
This makes the overall time complexity to be $O(n \log n)$.

4 Proof of Optimality

Our main aim was to minimize the maximum of annoyance among all the professors. Let us assume that the algorithm defined above gives us a solution S and let O be the optimal solution with fewest number of *inversions* among all the optimal solutions.
Firstly, an *inversion* is defined as follows:
There are two classes i and j with $d_i < d_j$ and class j is scheduled before the class i.
According to the algorithm defined, S will have 0 *inversions.* If even O has 0 *inversions*, then our job of proving that the algorithm is *optimal* is done. Hence, O must have atleast 1 *inversion.*
Now, we swap class i and class j. Let $annoyance_k$ be the annoyance of the professors before swapping the classes (where k $\in \{i, j\}$) and let $annoyance'_k$ be the annoyance of the professors after swapping.

Now, $annoyance'_j = f'_j - d_j = f_i - d_j$ (because of swapping, $f'_j = f_i$)
Since $d_i < d_j$,
$annoyance'_j <= f_i - d_i$
$annoyance'_j <= annoyance_i$
Hence, we saw that maximum annoyance does not increase after swapping a pair with inversion.

So if we swap the classes i and j in O, we do not change the annoyance but this will reduce the number of *inversions* by 1. But this contradicts our assumption that O has least number of *inversions*. Hence, we now know that there definitely exist an *optimal* solution with 0 *inversions*. And since our solution S also has 0 *inversions*, we can conclude that the solution obtained by the algorithm described above is *optimal*.

Therefore, the algorithm defined above (completing the classes based on increasing order of deadlines) by greedy approach is *optimal*.