Trevor Stahl

2/1/2019

HW4

CS 325 400 w2019

Problem 1

Suppose you have a set of classes to schedule among a large number of lecture halls, where any class can class place in any lecture hall. Each class cj has a start time sj and finish time fj. We wish to schedule all classes using as few lecture halls as possible. Verbally describe an efficient greedy algorithm to determine which class should use which lecture hall at any given time. What is the running time of your algorithm?

First sort the classes so that the class with the earliest start time is first. Take the next/first cj value and put it into the first lecture hall where its sj and fj do not overlap with the sj and fj of a cj that has been assigned to that lecture hall. If there are no current lectures halls to put it in where the start and finish times do not overlap then create a new lecture hall to use and put it there. Repeat this until there are no more classes to put into lecture halls.

Theta(n^2)

OR O(n^2) depending on exact implementation

Problem 2

For each $1 \leq i \leq n$ job ji is given by two numbers di and pi, where di is the deadline and pi is the penalty. The length of each job is equal to 1 minute and once the job starts it cannot be stopped until completed. We want to schedule all jobs, but only one job can run at any given time. If job i does not complete on or before its deadline, we will pay its penalty pi. Design a greedy algorithm to find a schedule such that all jobs are completed and the sum of all penalties is minimized. What is the running time of your algorithm?

Order jobs based upon penalty size such that the greatest penalty is dealt with first. Schedule the jobs by taking the first/next job to be scheduled and placing into the schedule at the latest slot possible to finish it before deadline. If there are no available slots between now and when it is due allow the penalty to be taken and do the job after the last deadline.

n log(n) + n(n/2)

n log(n) + n^2

O(n^2)

Possible that it is sorted on penalty already and possible that the first place we look is unoccupied every time. So just Big Oh not theta.

Problem 3

Activity selection first-to-finish is a greedy algorithm because it always chooses the local optimal by adding to the array of activities to be done by adding the first activity to finish. This is optimal because there will never be a scenario where doing otherwise would be better. It cannot be better because if we choose the one that finished later than another less time will be available to fill up with more activities. Similarly, Activity Selection Last-to-Start is also greedy because it adds to the array of activities to ideally complete by adding the activity with the latest start time. This is similarly optimal because if we ever chose an activity that started earlier than another there would be less time for earlier activities.

Compared to Activity selection first-to-finish, Activity Selection Last-to-Start is basically the same procedure but with reversed direction. We proved above that it is optimal.

Problem 4

Verbal Description: Last to start is an algorithm that takes input as a list of activities with start and finish times. The algorithm sorts according to latest start time first. And then it produces a second list which will be the result, by adding to the list the activity that starts last and then adding the next activity that starts last but does not overlap with what is already in results list.

Pseudo-Code:

```
Sort activities from latest start time first to earliest start time last
    Set n to the number of activities in list
    activities_result = list with only first activity in it
    k = 0
    for m=2 to n
       if finish time of act m <= start time of act k
          add to activities_result the id number of act m
          k = m
    reverse the order of activities_result
```

Theoretic Running Time: O(n*lgn)

The most complex step in code is sorting. Sorting is (n lgn). Could have something that is sorted already so not theta but Big Oh.