CS 325 Spring 2018 – HW 4

Sheng Bian

Problem 1:

Machine Scheduling can be used for this problem. We created an algorithm called ClassSchedule(C). The input is set C of classes with start times s_j and finish time f_j. The output is non-conflicting schedule with minimum number of lecture halls. The algorithm ClassSchedule(C) is described below.

```
h = 0 {number of lecture halls}
while C is not empty                            (Running time is O(n))
    remove class j with smallest $s_j$
    if there’s a lecture hall i for j then      (Running time is O(logn)
        schedule j on lecture hall i
    else
        h = h + 1
        schedule j on lecture hall h
```

Therefore, the running time of my algorithm is O(n logn).

Problem 2:

In order to minimize the number of days it takes get to destination, we should drive to the farthest hotel within d miles and rest there for one night. Keep doing that until we get to the last hotel.

The running time of this algorithm O(n) since we need to go through each hotel sequentially.

Problem 3:

Assume T_i is the time interval to complete each job, $1 \leqslant i \leqslant n$, and T_i starts from i-1 and ends at i. The greedy algorithm is described as follows.

Sort the jobs in decreasing order of penalties $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$

```
if these is time interval $T_m$ available for $1 \leqslant m \leqslant d_i$
        schedule $j_i$ in the last available time interval
else
        schedule $j_i$ in the first available time interval after $M_n$
```

Running time of algorithm is $O(n^2)$. The reason is sort the jobs will take time O(n logn) and find the correct spot would take time $O(n^2)$. Therefore, the running time is $O(n^2)$.

Problem 4:

This approach is a greedy algorithm and it starts from the end rather than from the beginning.

Prove: There is a set of activities S = $\{a_1, a_2, a_3, \ldots, a_n\}$ where the starting time of a_i is s_i and the finishing of a_i is f_i. Create a set S' = $\{a_1', a_2', a_3', \ldots, a_n'\}$ where the starting time of a_i' is f_i and the finishing of a_i is s_i. a_i' is the reverse of a_i. Then, a subset of $\{a_1, a_2, a_3, \ldots, a_n\} \subset S$ is mutually compatible if and only if $\{a_1', a_2', a_3', \ldots, a_n'\} \subset S'$ is mutually compatible. Therefore, the optimal solution for S is directly mapping to optimal solution for S'.

We can see the approach is exact the time but perform the algorithm in the opposite direction of the set. Thus, the approach to find optimal solution for S corresponds to the approach to find optimal solution for S'. Therefore, it yields an optimal solution.

Problem 5:

The following is the verbal description of your algorithm:

1) Sort the activities according to their starting time
2) Select the first element and append it to the selectedActivities
3) Do following for remaining activities in the sorted array: If the finishing time of this activity is less than or equal to the starting time of previously selected activity then select this activity and append it to the selectedActivities

The following is pseudocode:

Assume the activities sorted by starting time   (running time n logn)

```
Last-To-Start-Greedy-Activity-Selector (s, f)
    n <- length[s]
    A <- {$a_1$}
    i <- 1
    for m <- 2 to n                                   (running time n)
        do if $f_m <= s_i$
            then A <- A U {$a_m$}
                i <- m
    return A
```

The theoretical running time is O(n logn)