Bryan Rodriguez

CS 325 Fall 2020

HW 4

Problem 1

- Describe an efficient greedy algorithm that determines which hotels you should stay in if you want to minimize the number of days it takes you to get to your destination.

  Algorithm:

  For each day of travel:

  a) Iterate through hotel distances comparing their distance to the drive distance "d"
     a. Once the hotel distance is greater than the "d", take the previous hotel's distance as the hotel to stay at for the night.
     b. This hotel's location is the new start point for the next day
     c. Repeat this loop until the last hotel is reached.
- What is the running time of your algorithm?

  Run time of this would be O(n)

Problem 2

- Design a greedy algorithm to find a schedule such that all jobs are completed, and the sum of all penalties is minimized.

  Known:
  a. job j_i is given by two numbers d_i and p_i
  b. d_i = deadline
  c. p_i = penalty
  d. restrictions:
     i. each job length == 1 minute
     ii. once the job start it cannot be stopped until completed'
     iii. only one job can be run at a time
     iv. if job I does not complete before deadline, a penalty p_i is paid.

  Algorithm:

  a) the penalties for the jobs need to be sorted in descending order
  b) if the deadline of the job is equal to i or i is equal to n
     a. schedule the job in the time slot
  c) else, continue to call the algorithm on the rest of the jobs.
- What is the running time of your algorithm?

  Run time here would be the sort + the algorithm, so O(nlogn)

Problem 3

- Describe how this approach is a greedy algorithm and prove that it yields an optimal solution.

  This implementation of a greedy algorithm would now start at the end of the process rather than the start. It is greedy because it is still making the optimal choice, here selecting the last activity to start so it follows the given restrictions. From 16.1-1, the recurrence relation used from 16.2 yields an algorithm that that selects for an activity that is first to finish. If this algorithm is running in reverse, then it will produce the optimal solution so long as it follows the greedy method given from its constraint of selecting an activity that is compatible with all previously selected activities. Proving that it will yield the optimal solution because it makes a greedy choice at each iteration.

Problem 4

- This algorithm takes the job data from act.txt, sort the job data, then pass it to the greedy algorithm in reverse order. This would mean that the job that ends the latest is always taken first, the algorithm then compares the first job's starting time with the latest start time available for the next job. This is the greedy step always taking the last activity to start that is compatible with all previously selected activities.
- This pseudo code looks like this:
    - **a.** Parse data act.txt file
    - **b.** Append each job consisting of the job number, start time, and end time as a list of lists into a jobs[] list
    - **c.** jobs.sort(sort by the end times of jobs, reversed)
    - **d.** pass this jobs[] list into the greedy algorithm
        - **i.** greedyalgorithm(array)
            1. create a new jobs list jobs[]
            2. for job in jobs list:
                - **a.** if the length of jobs == 0
                    - **i.** append jobs with the first job in the passed array
                - **b.** elif array[job][endtime] <= jobs[-1][starttime]:
                    - **i.** append jobs with array[jobs]
            3. jobs_taken = [a new list from the completed jobs list]
            4. reverse(jobs_taken)
            5. return(jobs_taken)
    - **e.** Print out desired output with results from greedy algorithm
- Runtime of this algorithm would be O(nlogn)

Activity.py ReadMe:

1. Open the activity.py file and make sure your act.txt file is in the proper directory for it to be imported.
2. The activity.py file will output its results directly to the terminal
3. The terminal will display the results in this format

    Set [n]
    Number of activites = [k]
    Activites: {j_1, j_2…j_k}