Class: CS-325
Term: Fall 2017
Author: Jon-Eric Cook
Date: October 22, 2017
Homework: #4

Problem 1: (5 points) Class Scheduling:

Suppose you have a set of classes to schedule among a large number of lecture halls, where any class can be placed in any lecture hall. Each class c_j has a start time s_j and finish time f_j. 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?

Answer:

A greedy algorithm that is both efficient and can determine which class should use which lecture hall at any given time is one that considers classes by their start times. The algorithm should first sort the classes in ascending order based on their start times and start out with zero lecture halls. The algorithm should select the class with the earliest start time, the first element in the list, and create a lecture hall for it. The selected class should be scheduled in newly created lecture hall and removed from the list. It should then select the next class in the sorted list and check if its start time conflicts with the end time of the previous class. If there is a conflict, then a new lecture hall is to be created and the class scheduled in it and removed from the list. But if not, then the selected class can be placed in the previously created lecture hall and removed from the list. These steps are to be followed until there are no more classes in the sorted list. The running time of my algorithm is theta(nlgn).

Problem 2: (5 points) Road Trip:

Suppose you are going on a road trip with friends. Unfortunately, your headlights are broken, so you can only drive in the daytime. Therefore, on any given day you can drive no more than d miles. You have a map with n different hotels and the distances from your start point to each hotel $x_1 < x_2 < ... < x_n$. Your final destination is the last hotel. 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. What is the running time of your algorithm?

Answer:

A greedy algorithm that selects a hotel with a distance that maximizes the allowable distance traveled in a day is one that is efficient at determining which hotels should be stayed in to minimize the number of days it would take to get to the destination. An assumption is made that the provided hotel distance array is already sorted in ascending order. The algorithm would then calculate the current distance traveled from the starting point and then select a hotel by stepping through the sorted list of hotels and checking if its distance, when the current distance is subtracted form it, is equal to or less than the allowable travel distance in a day. This should be repeated until arriving at the final destination hotel. The running time for my algorithm is O(n).

Problem 3: (5 points) Scheduling jobs with penalties:

For each $1 \leq i \leq n$ job j_i is given by two numbers d_i and p_i, where d_i is the deadline and p_i 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 p_i. 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?

Answer:

A greedy algorithm that favors penalties can find a schedule such that all jobs are completed and the sum of all penalties is minimized. First, the jobs array would be sorted into descending order, based on the penalty value. From here, the job with the highest penalty would be scheduled such that it would run and complete right one its deadline, so as to avoid the penalty. This procedure would be followed for all the jobs in the descending list. When a job needs to be scheduled in an slot that is taken, it will be forced to be scheduled in the next available slot, after its deadline, thus incurring a penalty. The running time for my algorithm is O(n^2).

Problem 4: (5 points) CLRS 16.1-2 Activity Selection Last-to-Start

Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

Answer:

The original approach, selecting the first activity to finish, builds a gantt chart from the top left down to the bottom right, with no overlap. This is done by selecting the activity that finishes first and placing it on the gantt chart, in the top left most position. The second activity is selected such that it is the first activity to start after the finish of the previous activity. This selected activity is then place on the gantt chart, below the previous activity. This pattern is followed until there are no more activities to choose from. For the proposed approach, this exact same procedure is used but in reverse… moving from the top right, down to the bottom left of the gantt chart.

The original approach does comparisons between the finish time of the previously selected activity and the start time of the next to be selected activity. The proposed approach does comparisons between the start time of the previously selected activity and the finish time of the next to be selected activity. The comparisons are the same, but merely flipped. Thus, the proposed approach of selecting the last activity to start that is compatible with all previously selected activities yields an optimal solution.

Simply put, the original approach finds the optimal solution for a set S={a1,a2,…an} of activities, where ai=[si,fi]. The proposed approach needs to find the optimal solution for a set S`={a1`,a2`,…an`} of activities, where ai`=[fi,si]. ai` is the reverse of ai. ai is subset of S so then ai` is a subset of S`. Because of this, an optimal solution for S maps to an optimal solutions for S` and vice versa.

Problem 5: (10 points) Activity Selection Last-to-Start Implementation

Submit a copy of all your files including the txt files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named act.txt.

You may use any language you choose to implement the activity selection last-to-start algorithm described in problem 4. Include a verbal description of your algorithm, pseudocode and analysis of the theoretical running time. You do not need to collected experimental running times.

The program should read input from a file named “act.txt”. The file contains lists of activity sets with number of activities in the set in the first line followed by lines containing the activity number, start time & finish time.

Example act.txt:

11

1 1 4

2 3 5

3 0 6

4 5 7

5 3 9

6 5 9

7 6 10

8 8 11

9 8 12

10 2 14

11 12 16

3

3 1 8

2 1 2

1 3 9

In the above example the first activity set contains 11 activities with activity 1 starting at time 1 and finishing at time 4, activity 2 starting at time 3 and finishing at time 5, etc.. The second activity set contains 3 activities with activity 3 starting at time 1 and finishing at time 8 etc. Note: the activities in the file are not in any sorted order.

Your results including the number of activities selected and their order should be outputted to the terminal. For the above example the results are:

Set 1
Number of activities selected = 4

Activities: 2 4 8 11

Set 2
Number of activities selected = 2

Activities: 2 1

Answer:

Verbal description

My algorithm has three major portions. The first portion opens the text file and retrieves the data for a set of activities. Only the data for the fist set of activities is obtained and processed into a list of lists. The next portion sorts the list of lists into descending order based on the start times. This results in a list of lists that starts with the activity with the largest start time. The next portion walks through the sorted list of lists and selects the most activities that do not conflict with each other. Lastly, the results of the selected activities is printed to the terminal. This process is repeated until the end of the input text file is reached.

Pseudocode

```
def main()
        open the act.txt file
        while True
                get a line from the text file
                if the line is empty
                        break
                convert line to list of integers
                if length of line == 1
                        set counter to line[0]
                for i from 0 to counter
                        get a line from text file
                        convert line to list of integers
```

put into an list

sorted_activities = sort(list of lists of activities)

select_activities = select(sorted_activities)

print the set number

print the number of activities selected

print the selected activities

Theoretical Running Time

O(nlgn)

I believe this to be the theoretical running time because the longest operation is the merge sort of the list of lists. The other time consuming operation is a simple for loop that runs for the number of activities in a given set.

For code, see .zip file submitted to TEACH.