CS 477: Homework #6

Due on November 17th, 2016 at 2:30pm

Monica Nicolescu

Matthew J. Berger

Problem 1

1.) [40 points](U & G Required)

A company with a photocopying service that has a single large machine faces the following scheduling problem. Every morning, the company receives a set of jobs from its customers. Ideally, the jobs should be run on their single machine in an order that keeps their customers happiest. We know that customer i's job will take t_i time to complete. Given a schedule (i.e., an ordering of the jobs), let C_i represent the finishing time of job i. For example, if job i is the first to be done, we would have $C_i = t_i$; and if job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. Therefore, the company decides that they want to order the jobs to minimize the weighted sum of the completion times $\sum_{i=1}^{n} w_i C_i$.

Design an efficient algorithm to solve this problem: given a set of n jobs with a processing time t_i and a weight w_i for each job, order the jobs so as to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.

Example: Suppose there are two jobs: the first takes time $t_1 = 1$ and has weight $w_1 = 10$, while the second job takes time $t_2 = 3$ and has weight $w_2 = 2$. Then doing job 1 first would yield a weighted completion time of $10 * 1 + 2 * 4 = 18$, while doing the second job first would yield the larger weighted completion time of $10 * 4 + 2 * 3 = 46$. Note: you have to prove that your strategy yields the optimal solution.

Solution

For a set of jobs, each job i has been assigned a weight and a processing time. After sorting S we'll have a completion time C_i for each job i. The greedy solution is to do the job with the larger weight/time first because in unit time the job has the larger weight, minimizing our formula.

1: **procedure** MINIMIZEWSCT(a, b) ▷ Minimize Weighted Sum of Completion Times
2: *ratioList* $\leftarrow <$ *value of weight/time for each job* $>$
3: Sort the ratioList in descending order. ▷ Process the jobs in this order!
4: **end procedure**

Algorithm 1: MinimizeWSCT

Problem 2

2.) [40 points](U & G Required)

A student camp counselor is in charge of organizing activities for a set of junior-highschool-age campers. He plans the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles. His plan is to send the contestants out in a staggered fashion, based on the following rule: the contestants must use the pool one at a time. In other words, the first contestant swims the 20 laps, gets out, and starts biking. As soon as the first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming and so on.). Each contestant has a projected *swimming time* (the expected time it will take him or her to complete the 20 laps), a projected *biking time* (the expected time it will take him or her to complete the 10 miles of bicycling), and a projected *running time* (the time it will take him or her to complete the 3 miles of running). The camp counselor wants to decide on a *schedule* for the triathlon: an order in which to sequence the starts of the contestants. Lets say that the completion time of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathlon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. (Again, note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.) Whats the best order for sending people out, if one wants the whole competition to be over as early as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible. Note: you have to prove that your strategy yields the optimal solution.

Problem 3

3.) [20 points](U & G Required)

Construct an optimal Huffman coding tree for the characters I with frequency 75, U with frequency 200, B with frequency 25, S with frequency 275, C with frequency 50, H with frequency 100, M with frequency 25, and P with frequency 250. How many bits are required to store the above characters using the resulting encoding? (Show your work for building the Huffman codes).

Problem 5

5.) [20 points](Extra Credit)

Consider a long, quiet country road with houses scattered very sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint). The residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient (greedy) algorithm that achieves this goal, using as few base stations as possible. Note: you have to prove that your strategy yields the optimal solution.

Solution

A possible greedy algorithm to solve this would be one that starts from the leftmost point and at each step places the beginning of the interval at the first uncovered point. This is equivalent to covering a set of points on a real line with 8-unit length intervals, with the goal being to use the minimum number of intervals.

To prove this, we can use contradiction. Greedy is not optimal. Let $g_1 < g_2 < ... < g_n$ represent the set of starting positions indicated by the intervals chosen in the algorithm. Let $f_1 < f_2 < ... < f_n$ denote the set of starting points at the intervals of the optimal solution with $f_1 = g_1, f_2 = g_2, \ldots, f_n = g_n$ for the largest possible value of n. $g_{n+1} > f_{n+1}$ because of the greedy algorithm. The optimal solution is modified further by making $f'_{n+1} = g_{n+1}$ (pushing the $(r+1)_{th}$ interval further). This covers all of the points, because there cannot be any points in (f_{n+1}, g_{n+1}) due to the greedy algorithm. The number of intervals does not change, therefore it is optimal. The contradiction is that the new optimal solution has one more starting point in common with the greedy solution.