Foundations of Algorithms, Spring 2021: Homework 4

Due: Wednesday, March 31, 11:59pm

Problem 1 (12 points: 8 for implementation / 4 for writeup)

Consider the following variant of the interval scheduling problem. You are a contractor and want to complete as many jobs (intervals) as you can. You do work for two different employers. In order to keep both employers happy, you must alternate doing jobs for the employers. You can choose either employer for the first job, but must make sure never to do two consecutive jobs for the same employer.

Given are n intervals (jobs). The i^{th} interval is specified by (s_i, f_i, e_i), where the interval runs from starting time s_i to finishing time f_i, and the employer is e_i. Design an $O(n \log n)$ algorithm that determines the maximum number of compatible jobs that you can complete while keeping both employers satisfied (i.e. while alternating employers). Note that if job j has a finish time that is equal to the start time of job k, jobs j and k are considered compatible.

Problem 2 (20 points: 14 for implementation / 6 for writeup)

Given is a sequence of integers: $X = \{x_1, x_2, \ldots, x_n\}$.

a) You are interested in partitioning the sequence X into non-empty consecutive subsequences such that the sum of each subsequence is even. A consecutive subsequence is defined as a subsequence in which all of the chosen elements are consecutive in the original sequence.

   For example, given $X = \{3, 6, 1, 2, 4\}$, one solution is to divide X into the consecutive subsequences $\{3, 6, 1\}$ and $\{2, 4\}$, having even sums 10 and 6, respectively.

   Design a greedy $O(n)$ algorithm that computes the number of unique ways the sequence X can be partitioned into non-empty consecutive subsequences such that the sum of each subsequence is even.

   In the example above, the answer is 4. In addition to the partition shown, there are three other possible partitions: $[\{3, 6, 1, 2\}, \{4\}]$, $[\{3, 6, 1\}, \{2\}, \{4\}]$, and $[\{3, 6, 1, 2, 4\}]$ (the entire sequence).

b) Now you are interested in a very similar problem. You want to partition the sequence X into consecutive subsequences such that the sum of each subsequence is odd.

   For example, given $X = \{3, 6, 1, 2, 4\}$, one solution is to divide X into the consecutive subsequences $\{3, 6\}$ and $\{1, 2, 4\}$, having odd sums 9 and 7, respectively.

   Design an $O(n^2)$ dynamic programming algorithm that computes the number of unique ways the sequence X can be partitioned into consecutive subsequences such that the sum of each subsequence is odd.

   In the example above, the answer is 2. In addition to the partition shown, there is one other possible partition: $\{3\}, \{6, 1, 2, 4\}$.

c) BONUS (3 points): Design a dynamic programming algorithm for part b) that runs in $O(n)$ time. Note that a solution to the bonus also satisfies the requirements for b).

Problem 3 (10 points: 7 for implementation / 3 for writeup)

Given is a sequence of integers, $a_1, a_2, \ldots, a_n$. Give an $O(n^2)$ algorithm that finds the largest possible sum of elements in an increasing subsequence of $a_1, a_2, \ldots, a_n$.

Problem 4 (18 points: 12 for implementation / 6 for writeup)

You are working at an amusement park on a day when a large company reserves the entire park for its employees. You are working at a ride fed by two lines. You already feel a headache coming on, and your goal is to get everybody in the two lines onto the ride with minimum aggravation of your headache. Here are the characteristics of your problem:

1. There are three types of employees: (E)xecutives, (V)eterans, and (N)ew Hires.
2. The two lines are filled with a mix of the three different types of employees. Line 1 has m people in it. Line 2 has n people in it.
3. The ride fits two people at a time. Standard procedure is to take the person at the front of each line and put them together on the ride. However...
4. E's and N's don't want to sit together. Matching an E with an N causes grumbling that incurs 5 units of headache for you. (Any other pairing causes no problem.)
5. You can choose to fill a ride by taking two people from the front of the same line. However, this causes grumbling from the other line that incurs 3 units of headache to you, unless the other line is empty. If the two people you take from the front of the same line happen to be an E and an N, you would also incur an additional 5 units of headache (see above).
6. You can also choose to fill a ride with only one person. But then everybody remaining complains about the inefficiency, incurring 4 units of headache, unless there is nobody left at that point in either line (this is the last person to ride).

For example, suppose Line 1 = $\{E, E, N\}$, and Line 2 = $\{N, N, V, E\}$, where the lines are listed from the back to the front. You could start by pairing V, E from line 2 at a cost of 3. Then pair N from the front of each line at no cost. Then match E, E from line 1 at a cost of 3. Finally, N from line 2 rides alone at no cost (because there is nobody left). Thus the total cost is 6. However, this is not optimal. You could do better by letting N from line 1 ride alone to start, at a cost of 4. Then pair the front of each line twice (E, E and E, V) at no cost. Finally, only N, N remains in line 2, and can ride at no cost (since the other line is empty). Thus the total cost is 4. No other solution yields a total less than 4.

Give an $O(mn)$ dynamic programming solution to this problem, that computes the minimum units of headache you will incur getting all of the company employees on the ride.