CSE100 Algorithm Design and Analysis Discussion. Chapter 16

You're expected to work on the discussion problems before coming to the lab. Discussion session is not meant to be a lecture. TA will guide the discussion and correct your solutions if needed. We will not release 'official' solutions. If you're better prepared for discussion, you will learn more. TAs will record names of the students who actively engage in discussion and report them to the instructor; they are also allowed to give some extra points to those students at their discretion. The instructor will factor in participation in final grade.

1. (Intermediate) In the Interval Selection (Activity Selection) problem, we're given n interval $I_i = (s_i, f_i)$, $1 \leq i \leq n$ and would like to choose the maximum number of mutually disjoint/non-overlapping intervals. Explain why the greedy algorithm that recursively chooses the earliest ending interval (and discards overlapping intervals) is optimal.

   (a) Why is the following claim true? Claim: There is an optimal solution that includes I_1 which ends the earliest.
   Sol. Omitted.

   (b) Explain why the claim implies the earliest finishing algorithm is optimal.
   Sol. Omitted.

2. (Basic) What is fixed-length code? What is prefix (or prefix-free) code? Is a fixed-length code always prefix-free? Why do you need prefix-freeness?

   Sol. Fixed-length code: all codewords are of an equal length. We say that a code is prefix-free if no codeword is a prefix of other codewords. Yes, a fixed-length code is always prefix-free. We need prefix-freeness since otherwise decoding is not unique.

3. (Basic) Assume the following when executing the Huffman algorithm: When combining two trees, the tree with lowest root frequency becomes the left child and the tree with the second-lowest root frequency becomes the right child. Left children are associated with the bit 0, right children with the bit 1.Run the Huffman's algorithm on the following input: $a : 3$, $b : 2$, $c : 4$, $d : 8$, $e : 7$, $f : 14$. What is the codeword for each character? Give a tree representation.

   Sol.

   a:1011
   b:1010
   c:100
   d:01
   e:00
   f:11

   For a tree representation see the textbook/lecture slides: 'left' edges are tagged by 0 and 'right' edges by 1.

4. (Basic) Repeat the previous question on the input, $a : 1$, $b : 2$, $c : 4$, $d : 5$, $e : 6$, $f : 9$.

   Sol. Omitted.

5. (Basic) Explain why a prefix-free code is not optimal if some node has exactly one child in the tree representation.

   Sol. Omitted.

6. (Advanced) Single Machine Scheduling + Completion Time Minimization. We are given n jobs, $1, 2, ..., n$ with sizes $p_1, p_2, ...p_n$. To complete job i we need to process job i for p_i units of time. We have a single processor and can start processing jobs from time 0. At each time, we can process at most one job. Our goal is to find a schedule that minimizes the total completion time. Show an optimal algorithm and prove the optimality. For simplicity, you may assume that jobs must be scheduled non-preemptively, meaning that once you start processing a job, you have to finish it before you process other jobs.

   Example. Say that we have two jobs with sizes $p_1 = 1$ and $p_2 = 2$. If you process job 1 first and then job 2, you can complete jobs 1 and 2 at times 1 and 3, resp., thus having total completion time 4. On the other hand, if you finish job 2 and then job 1, then you'll complete jobs 1 and 2 at times, 3 and 2, resp., thus having total completion time 5. So the former schedule is better.

   Sol. In words, we would like to find an ordering of jobs such that the total waiting time of all jobs is minimized. We show that the shortest job first algorithm is optimal; you can view the algorithm of recursively choosing the shortest job among the remaining one. The key lemma we want to show is that there is an optimal solution that schedules the shortest job first. If we show the lemma, we can recursively apply the lemma: we know that we can safely finish the shortest job first. Then, how do we minimize the total waiting time of other jobs? (they are delayed by a length of time equal to the first job's size). By applying the lemma again, we can safely process the shortest remaining job. That is, we can schedule jobs in increasing order of their job sizes, and the resulting schedule will be optimal.

   Now it remains to show the lemma. Consider any optimal schedule. If the optimal schedule schedules the shortest job, say j, first, then we are done. So, say that the schedule schedules job i first instead of j. Swap the two jobs i and j. Then we can show that every job's completion time can only decrease, meaning that we got another optimal schedule scheduling the shortest job first.

7. (Advanced) Single Machine Scheduling Meeting Jobs Deadlines. As before, we have n jobs, $1, 2, ..., n$ with sizes $p_1, p_2, ...p_n$. Here each job j also has a deadline d_j, meaning that we are required to complete the job by time d_j. We want to check if there is a schedule that completes all jobs by their respective deadlines. Show that scheduling the job with the earliest deadline (a.k.a. Earliest Deadline First (EDF)) finds such a schedule if it exists. As before, the scheduler can start processing any job from time 0.

   * Note: This holds true even if jobs arrive over time. But for simplicity, let's assume that all jobs are available for schedule from the beginning, time 0.

Sol. The key lemma: there is a feasible (in the sense that all jobs deadlines are met) schedule where the job j with the latest deadline completes the latest.

As before, once we have this lemma, by recursively applying the key lemma, we can conclude that EDF is optimal.

Now let's focus on showing the lemma. Consider any feasible schedule where all jobs complete before their deadlines. Suppose the schedule completes another job i the latest since otherwise we already have the lemma. Swap j and i. Then, it is easy to see that all jobs except job j complete only earlier. So, they still must meet their deadlines in the new schedule. Now let's shift out attention to job j. Note that j's completion time in the new schedule is exactly the same as i's completion time in the old schedule. But in the old schedule, job i completed before its deadline, which is only smaller than j's deadline. Hence j completes before its deadline in the new schedule.