Solutions to Practice Problems 3

Solution 1: The solution is shown in Fig. 1, using a total of three colors.

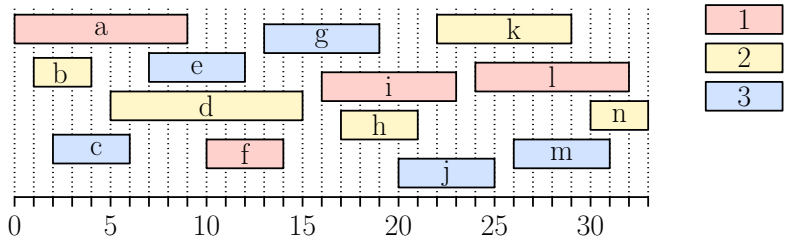

Figure 1: Interval partitioning.

Solution 2:

(a) To create a counterexample, we create one very long request with the earliest start time, and a set of $n-1$ pairwise disjoint requests that overlap the first request (see Fig. 2(a)). The ESF strategy will succeed in scheduling only the first request, while the optimum schedules the remaining $n-1$. Thus, the performance ratio (that is, the ratio between the optimum and greedy) is $n - 1/1 = n - 1$. This can be made arbitrarily large by increasing n.

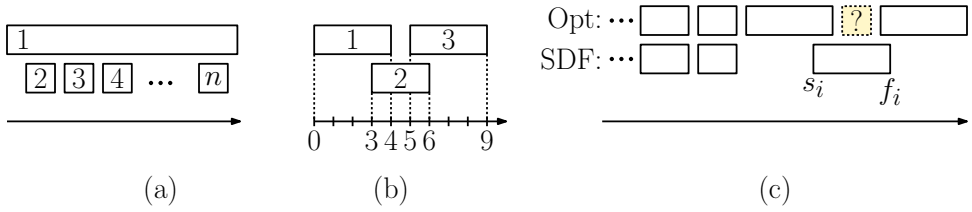

Figure 2: Interval scheduling.

(b) First observe that there is no loss of optimality by eliminating nesting. If $[s_i, f_i] \subset [s_j, f_j]$, then any schedule that uses $[s_j, f_j]$ cannot also use $[s_i, f_i]$, because they overlap. This means that we can replace $[s_j, f_j]$ with $[s_i, f_i]$ in any schedule without inducing any other overlaps. By repeating this, we convert any schedule to a nesting-free schedule without decreasing the number of scheduled requests. Therefore, we may assume that optimum schedule is nesting-free.

We assert that once nested intervals have been removed, sorting by start time is equivalent to sorting by finish time. To see why, suppose that $s_j \leq s_i$. By nesting-freeness, $f_j \leq f_i$, since otherwise we would have $s_j \leq s_i < f_i < f_j$, implying that $[s_i, f_i] \subset [s_j, f_j]$, a contradiction. Since EFF (earliest finish first) is known optimal for *any* set of requests, it follows that ESF is optimal on any set of nested-free intervals. Therefore ESF* is optimal.

(c) As a counterexample, consider three requests $I = \{[0,4],[3,6],[5,9]\}$ (see Fig. 2(b)). Requests 1 and 3 have duration 4 are non-overlapping, while request 2 has duration 3. Thus, SDF schedules only request 2, while the optimum schedules both 1 and 3. Therefore, $\text{Opt}(I) = 2$, but SDF(I) = 1.

(d) As with the proof that EFF is optimal, we will employ an induction proof where we repeatedly modify any Opt schedule to match SDF. In that proof, we swapped one-for-one, implying that both solutions had the same size. Here, we will swap at most two-for-one, implying that the size of SDF is at least half as large as Opt.

Given any instance I, consider the optimum and SDF sizes, denoted $\text{Opt}(I)$ and SDF(I), respectively. If they are the same, then $\text{Opt}(I) = \text{SDF(I)}$, and hence the both have the same size. Otherwise, find the first interval (in start-time order) such that $[s_i, f_i]$ is in the SDF solution but not in the Opt solution. We assert that at most two intervals from the Opt solution can overlap this interval. To see why, suppose to the contrary that three or more intervals of Opt were to overlap $[s_i, f_i]$. Then any interval other than the first and last would be completely contained within $[s_i, f_i]$, implying that this middle interval has a strictly smaller duration than $[s_i, f_i]$ (see Fig. 2(c)). Thus, SDF would have chosen this interval, rather than $[s_i, f_i]$, a contradiction.

Given the first difference $[s_i, f_i]$, we modify Opt by removing the (at most two) overlapping intervals from Opt and add $[s_i, f_i]$. The resulting schedule is clearly valid, and it has suffered a net decrease in size by at most one interval for each greedy interval on which we differ. By repeating this, we arrive at a schedule, denoted $\text{Opt}'(I)$ that is identical to greedy. The number of intervals that have been removed from the original optimum is not greater than the number of greedy intervals. Therefore, we have

$$\text{SDF}(I) \;=\; \text{Opt}'(I) \;\geq\; \text{Opt}(I) - \text{SDF}(I).$$

This implies that $2 \cdot \text{SDF}(I) \geq \text{Opt}(I)$, or equivalently, $\text{SDF(I)} \geq \text{Opt}(I)/2$, as desired.

Here is (arguably simpler) proof, which is based on a charging argument. Assign each interval of the optimum a *token*. The total number of tokens t is equal to $\text{Opt}(I)$. Whenever an interval of the optimum overlaps an interval of the SDF solution, transfer its token to the SDF interval. (If there are multiple such intervals in SDF, transfer the token to any one.) By the above observation, each interval of SDF receives at most two tokens. By adding up all the tokens in the SDF solution, we conclude that

$$\text{Opt}(I) \;=\; t \;\leq\; 2 \cdot \text{SDF}(I),$$

which implies that $\text{SDF(I)} \geq \text{Opt}(I)/2$, as desired.

Solution 3: Because the number of tasks is fixed, irrespective of which algorithm we use, we'll ignore the $(1/n)$ factor and compare schedules based on total lateness.

(a) Consider the tasks $(t_1, d_1) = (2, 0)$ and $(t_2, d_2) = (1, 1)$. If we use *earliest deadline first*, then $(s_1, f_1) = (0, 2)$ and $(s_2, f_2) = (2, 3)$, for a total lateness of $(2 - 0) + (3 - 1) = 4$. If we reverse the tasks, we have $(s_2, f_2) = (0, 1)$ and $(s_1, f_1) = (1, 3)$, for a total lateness of $(1 - 1) + (3 - 0) = 3$. Therefore, earliest deadline first is not optimal.

(b) Consider the tasks $(t_1, d_1) = (1, 3)$ and $(t_2, d_2) = (2, 2)$. If we use *smallest duration first*, then $(s_1, f_1) = (0, 1)$ and $(s_2, f_2) = (1, 3)$, for a total lateness of $0 + (3 - 2) = 1$. If we reverse the tasks, we have $(s_2, f_2) = (0, 2)$ and $(s_1, f_1) = (2, 3)$, for a total lateness of $(2-2)+(3-3) = 0$. Therefore, smallest duration first is not optimal.

(c) We will show that, using this modified definition of lateness, the *smallest duration first* greedy strategy minimizes total lateness. To prove this, we will employ the same sort of exchange argument that we used in class in proving the optimality of greedy algorithm for minimizing maximum lateness. As there, we may restrict attention to schedules that have no slack time. To simplify the proof, we will also assume that all the durations are distinct.

Consider a set $X = \{x_1, \ldots, x_n\}$ of tasks, let G be the schedule resulting from this strategy, and let O be an optimal schedule. We assert that $G = O$. Suppose not. Then, let x_i and x_j denote the first pair of tasks that violate the greedy principle, in the sense that $t_i > t_j$. The lateness of task i in schedule O will be denoted by ℓ_i^O and the lateness of task j in O will be denoted by ℓ_j^O (see Fig. 3). Similarly, let ℓ_i^G and ℓ_j^G denote the respective latenesses of tasks i and j in schedule G.

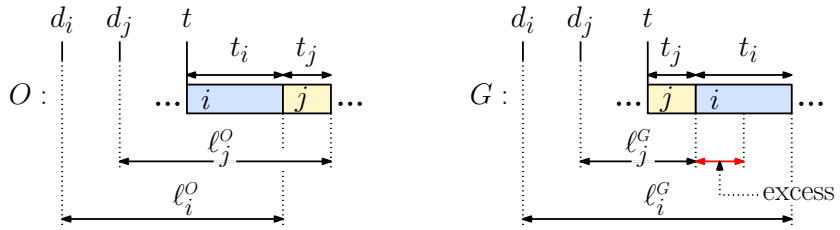

Figure 3: Solution to part (c).

Because the two schedules are identical up to these two tasks, and because there is no slack time in either, the first of the two tasks starts at the same time in both schedules. Let t denote this time (see Fig. 3). In schedule O, task i finishes at time $t + t_i$ and (because it needs to wait for task i to finish) task j finishes as time $t + (t_i + t_j)$. The lateness of each of these tasks are

$$\ell_i^O \;=\; t + t_i - d_i \qquad \text{and} \qquad \ell_j^O \;=\; t + (t_i + t_j) - d_j.$$

Applying a similar analysis to G, we can define the latenesses of tasks i and j in G as

$$\ell_i^G \;=\; t + (t_i + t_j) - d_i \qquad \text{and} \qquad \ell_j^G \;=\; t + t_j - d_j.$$

Given the individual latenesses, we can define their contribution to the total lateness of each schedule as their sum

$$L^O \;=\; \ell_i^O + \ell_j^O \qquad \text{and} \qquad L^G \;=\; \ell_i^G + \ell_j^G.$$

Our objective is to show that by swapping these two tasks, we decrease the total lateness. Since this in the only change, it suffices to show that $L^G < L^O$. By hypothesis, we have $t_i > t_j$. Therefore,

$$\begin{aligned} L^O - L^G &= (\ell_i^O + \ell_j^O) - (\ell_i^G + \ell_j^G) \\ &= (t + t_i - d_i) + (t + (t_i + t_j) - d_j) - (t + (t_i + t_j) - d_i) - (t + t_j - d_j) \\ &= t_i - t_j \;>\; 0, \end{aligned}$$

and therefore, $L^G < L^O$. But, since O is optimal, this is impossible. This yields the desired contradiction.

Solution 4: We present a simple greedy algorithm. The obvious greedy strategy is to go as far as possible before stopping to recharge. We maintain a variable `prev`, which indicates where we last recharged. The next charging station is the one that is farthest, but still within R miles (see Fig. 4(a)). The code block below provides a sketch of the algorithm. To avoid subscripting out of bounds, let us assume that $x[n+1] = x[n]$.

Greedy Algorithm for Recharging

```
getStops(x[0..n]) {
    prev = 0
    for (i = 1 to n) {
        if (x[i+1] > prev + R) {  // can't make it to x[i+1]?
            add i to the list of stops
            prev = x[i]             // save our last charge
        }
    }
}
```

Clearly the running time is $O(n)$. First, observe that greedy produces a feasible sequence, since we never go more than R miles before recharging. To establish optimality, let $O = \langle o_1, o_2, \ldots, o_k \rangle$ be the indices of an optimal sequence of recharging stops, and let $G = \langle g_1, g_2, \ldots, g_{k'} \rangle$ be the greedy sequence. (Note that by optimality, $k \leq k'$.) We will show that greedy is optimal by the usual replacement argument.

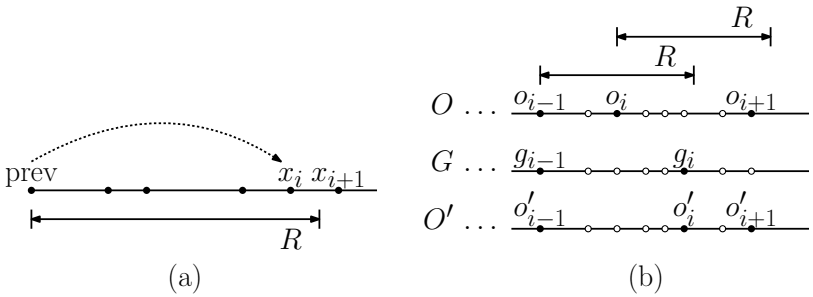

Figure 4: Greedy algorithm for recharging.

If the two sequences G and O are the same, then we are done. If not, let i be the smallest index where they differ (see Fig. 4(b)). Because greedy algorithm selects the *last* possible charging station, we know that $g_i > o_i$. (This also implies that $i \leq k$.) Consider an alternative solution O' which comes about taking O, but replacing o_i with g_i.

We claim that O' is a also a feasible solution. To see this observe that sequence up to g_i is the same as G (which we know is feasible). The only gap that has changed is the one between stops i and $i+1$. In O, it was of length $o_{i+1} - o_i$, and in O', it is of length $o_{i+1} - g_i$. Since $g_i > o_i$, the gap size has decreased. Therefore, the sequence is still feasible. The sequence O' has the same number of stops as O, and so it is also optimal. The number of differences between G and an optimal has

decreased by one. By repeating this, eventually we eventually transform the optimal solution into G, implying that G is optimal.

Solution 5:

(a) The counterexample involves two files, one slightly longer but with much higher access probability. Let $(s_1, p_1) = (1, 0.1)$ and $(s_2, p_2) = (2, 0.9)$. If we put f_1 before f_2 (size order), the expected access cost is $1 \cdot 0.1 + (2 + 1) \cdot 0.9 = 2.8$, but if we reverse the order of files the cost is $2 \cdot 0.9 + (1 + 2) \cdot 0.1 = 2.1$, which is smaller (see Fig. 5(a)).

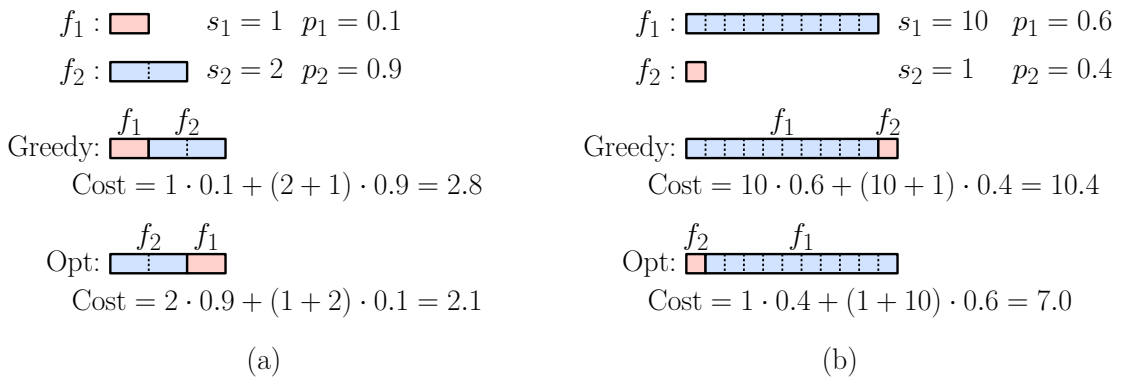

Figure 5: Optimal file layout.

(b) The counterexample involves two files, one slightly more likely to be accessed but with much larger size. Let $(s_1, p_1) = (10, 0.6)$ and $(s_2, p_2) = (1, 0.4)$. If we put f_1 before f_2 (decreasing probability order), the expected access cost is $10 \cdot 0.6 + (10 + 1) \cdot 0.4 = 10.4$, but if we reverse the order of files the cost is $1 \cdot 0.4 + (1 + 10) \cdot 0.6 = 7.0$, which is smaller (see Fig. 5(b)).

(c) Intuitively, it seems smart to store the most frequently accessed files at the front of the tape, but it also makes sense to store the smallest files at the front of the tape. This suggests that the best way to store the files is in increasing order of s_i/p_i. Let us sort the files according to this statistic and lay them out in this order. (We will make the simplifying assumption that these ratios are distinct for all files.) To simplify notation, let us assume that the files have been renumbered, so that $s_1/p_1 < \cdots < s_n/p_n$. Clearly, this layout can be computed in $O(n \log n)$ time.

We will prove that this is optimal by contradiction. Suppose that the optimum layout O is different from the greedy layout. If so, there must be two consecutive files of the optimum layout that are not in sorted order. That is, we have $O = \langle \ldots, f_j, f_i, \ldots \rangle$, where $j > i$ according to our greedy order. Thus, we have $\frac{s_j}{p_j} > \frac{s_i}{p_i}$, or equivalently (because sizes and probabilities are both nonnegative), $p_j s_i - p_i s_j < 0$.

Let us consider how the cost changes if these two files are swapped in the layout (see Fig. 6). Call the resulting layout O'. After the swap, file f_j has moved s_i units towards the back of the tape, and so its individual access cost has increased by $p_j s_i$. Similarly, file i has moved s_j units closer to the front of the tape, so its individual access cost has decreased by $p_i s_j$. All the other files maintain their same placements on the tape, so there are no other changes

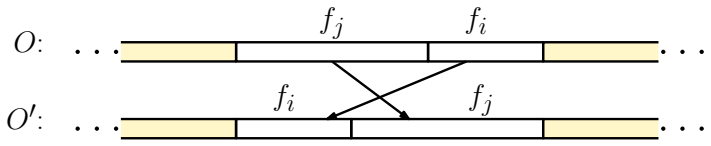

Figure 6: Optimality of the file layout solution, swapping f_j and f_i.

affecting the total cost. Therefore, the net change in the total access cost is:

$$T(O') - T(O) \;=\; p_j s_i - p_i s_j \;<\; 0.$$

Therefore, $T(O') < T(O)$, which contradicts the optimality of O, and yields the desired contradiction.