Solutions to Practice Problems 5

Solution 1:

(a) Interval $i < j$ is compatible with interval j exactly when $f_i \le s_j$. For each j, $p(j)$ is the largest $i < j$ with $f_i \le s_j$:

j	1	2	3	4	5	6	7	8
s_j	0	1	4	2	6	5	8	9
$p(j)$	0	0	1	0	3	2	4	5

For example, $p(5) = 3$ because $s_5 = 6$, and interval 3 is the last interval finishing by time 6 ($f_3 = 6$, $f_4 = 8$).

We use the recurrence from class,

$$\mathrm{OPT}(0) = 0, \qquad \mathrm{OPT}(j) = \max\big\{\, \mathrm{OPT}(j-1),\ \ w_j + \mathrm{OPT}(p(j)) \,\big\}.$$

j	$\mathrm{OPT}(j-1)$	$w_j + \mathrm{OPT}(p(j))$	$\mathrm{OPT}(j)$
0	—	—	0
1	0	$3 + \mathrm{OPT}(0) = 3$	3
2	3	$5 + \mathrm{OPT}(0) = 5$	5
3	5	$4 + \mathrm{OPT}(1) = 7$	7
4	7	$9 + \mathrm{OPT}(0) = 9$	9
5	9	$3 + \mathrm{OPT}(3) = 10$	10
6	10	$6 + \mathrm{OPT}(2) = 11$	11
7	11	$4 + \mathrm{OPT}(4) = 13$	13
8	13	$4 + \mathrm{OPT}(5) = 14$	14

The optimal total weight is $\mathrm{OPT}(8) = 14$.

Trace back. Start at $j = 8$. At each step, if $w_j + \mathrm{OPT}(p(j)) > \mathrm{OPT}(j-1)$, interval j is chosen and we move to $p(j)$; otherwise interval j is skipped and we move to $j - 1$.

- $j = 8$: $14 > 13$, so choose interval 8 and move to $p(8) = 5$.
- $j = 5$: $10 > 9$, so choose interval 5 and move to $p(5) = 3$.
- $j = 3$: $7 > 5$, so choose interval 3 and move to $p(3) = 1$.
- $j = 1$: $3 > 0$, so choose interval 1 and move to $p(1) = 0$, where we stop.

The optimal set is $\{1, 3, 5, 8\}$, i.e. the intervals $[0, 3], [4, 6], [6, 9], [9, 12]$, with total weight $3 + 4 + 3 + 4 = 14$. These are mutually compatible, as each one finishes no later than the next one starts.

(b) Give each interval the weight $w_j = f_j - s_j$, its length, and run the weighted interval scheduling algorithm. For any set of mutually compatible intervals, the total time scheduled is the sum of their lengths, which is exactly their total weight. So a maximum-weight compatible set is exactly a compatible set with the maximum total time.

(c) For $0 \le j \le n$ and $0 \le \ell \le k$,

$$\mathrm{OPT}(j,\ell) = \begin{cases} 0 & \text{if } j = 0 \text{ or } \ell = 0, \\ \max\{\, \mathrm{OPT}(j-1,\ell),\ \ w_j + \mathrm{OPT}(p(j),\ell-1)\,\} & \text{otherwise.} \end{cases}$$

The answer is $\mathrm{OPT}(n,k)$.

Explanation. The base cases hold because with no intervals available, or no intervals allowed, the best total weight is 0. For the recurrence, consider an optimal solution S for intervals $1,\ldots,j$ using at most ℓ intervals.

- If interval $j \notin S$, then S uses only intervals $1,\ldots,j-1$ and still at most ℓ of them, so its weight is $\mathrm{OPT}(j-1,\ell)$.
- If interval $j \in S$, every other interval in S is compatible with j. Since the intervals are sorted by finish time, the intervals among $1,\ldots,j-1$ that are compatible with j are exactly $1,\ldots,p(j)$. Interval j uses up one of the ℓ allowed intervals, so the rest of S is a solution for intervals $1,\ldots,p(j)$ with at most $\ell-1$ intervals, and it must be an optimal one (otherwise replacing it would improve S). Its weight is therefore $w_j + \mathrm{OPT}(p(j),\ell-1)$.

The optimal solution falls into one of the two cases, so $\mathrm{OPT}(j,\ell)$ is the larger of the two values. This is the recurrence from class with an extra index counting how many intervals we may still use.

Running time. Sorting and computing all $p(j)$ takes $O(n \log n)$. The table has $(n+1)(k+1)$ entries, and each is computed in $O(1)$ time in order of increasing j, since every entry depends only on entries with a smaller j. The total running time is $O(n \log n + nk)$. Since a solution never uses more than n intervals, we may assume $k \le n$, so this is at most $O(n^2)$.

Solution 2:

(a) We use the recurrence from class,

$$\mathrm{OPT}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \mathrm{OPT}(i-1,j-1) + 1 & \text{if } x_i = y_j, \\ \max\{\, \mathrm{OPT}(i-1,j),\ \ \mathrm{OPT}(i,j-1)\,\} & \text{if } x_i \ne y_j. \end{cases}$$

Filling in the table row by row gives the following, where row i corresponds to x_i in $X =$ `DYNAMIC` and column j corresponds to y_j in $Y =$ `DRAMATIC`. The bold entries are the matches used in the trace back below.

		D	R	A	M	A	T	I	C
	0	0	0	0	0	0	0	0	0
D	0	**1**	1	1	1	1	1	1	1
Y	0	1	1	1	1	1	1	1	1
N	0	1	1	1	1	1	1	1	1
A	0	1	1	**2**	2	2	2	2	2
M	0	1	1	2	**3**	3	3	3	3
I	0	1	1	2	3	3	3	**4**	4
C	0	1	1	2	3	3	3	4	**5**

The length of an LCS is $\mathrm{OPT}(7, 8) = 5$.

Trace back. Start at $(i, j) = (7, 8)$. If $x_i = y_j$, the character x_i is part of the LCS and we move to $(i-1, j-1)$. Otherwise we move to whichever of $(i-1, j)$ and $(i, j-1)$ has the larger value (going up on a tie). We stop when $i = 0$ or $j = 0$.

- $(7, 8)$: `C` $=$ `C`, take `C`, move to $(6, 7)$.
- $(6, 7)$: `I` $=$ `I`, take `I`, move to $(5, 6)$.
- $(5, 6)$: `M` $\neq$ `T`, and $\mathrm{OPT}(5, 5) = 3 > 2 = \mathrm{OPT}(4, 6)$, move to $(5, 5)$.
- $(5, 5)$: `M` $\neq$ `A`, and $\mathrm{OPT}(5, 4) = 3 > 2 = \mathrm{OPT}(4, 5)$, move to $(5, 4)$.
- $(5, 4)$: `M` $=$ `M`, take `M`, move to $(4, 3)$.
- $(4, 3)$: `A` $=$ `A`, take `A`, move to $(3, 2)$.
- $(3, 2)$: `N` $\neq$ `R`, tie $\mathrm{OPT}(2, 2) = \mathrm{OPT}(3, 1) = 1$, move up to $(2, 2)$.
- $(2, 2)$: `Y` $\neq$ `R`, tie $\mathrm{OPT}(1, 2) = \mathrm{OPT}(2, 1) = 1$, move up to $(1, 2)$.
- $(1, 2)$: `D` $\neq$ `R`, and $\mathrm{OPT}(1, 1) = 1 > 0 = \mathrm{OPT}(0, 2)$, move to $(1, 1)$.
- $(1, 1)$: `D` $=$ `D`, take `D`, move to $(0, 0)$ and stop.

The characters are collected from the end of the strings backwards, so reading them in reverse gives the LCS `DAMIC`. (In this example the LCS is unique, so breaking ties the other way leads to the same answer.)

(b) Let L be the length of the longest palindromic subsequence of X, and let C be the length of an LCS of X and X^R. We show $L \leq C$ and $C \leq L$.

$L \leq C$. Let P be a longest palindromic subsequence of X. Reversing X reverses each of its subsequences, so P^R is a subsequence of X^R. Since P is a palindrome, $P^R = P$, so P is a common subsequence of X and X^R. Hence $C \geq |P| = L$.

$C \leq L$. Let $Z = z_1 \cdots z_C$ be an LCS of X and X^R.

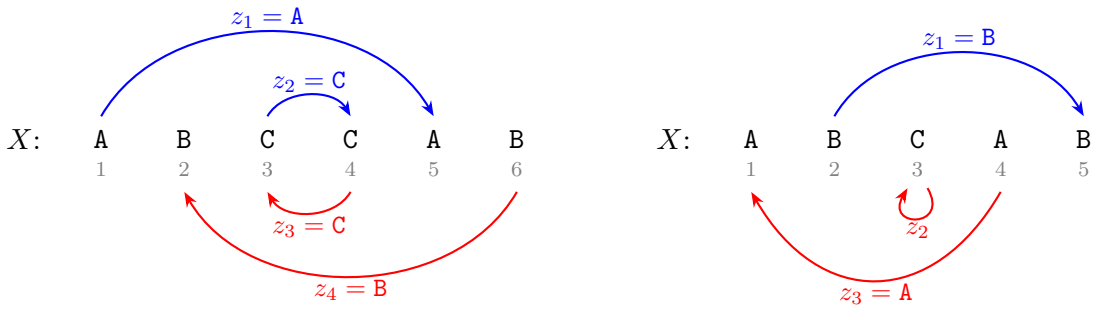
- Since Z is a subsequence of X, there are positions $i_1 < i_2 < \cdots < i_C$ in X with $x_{i_t} = z_t$.
- Since Z is a subsequence of X^R, and reading X^R from left to right is the same as reading X from right to left, there are positions $j_1 > j_2 > \cdots > j_C$ in X with $x_{j_t} = z_t$.

So every character z_t of Z appears twice in X: once at position i_t and once at position j_t. Draw an arrow from position i_t to position j_t for each t. As t increases, i_t moves right and

j_t moves left, so there is some k with $i_t < j_t$ for $t \le k$ and $i_t \ge j_t$ for $t > k$. In other words, the first k arrows point right and the remaining arrows point left (or have both ends at the same position). Within each group, every arrow lies inside the previous one, so the arrows are nested.

The figure below shows one example of each case.

- *Left ($k \ge C/2$).* $X = \texttt{ABCCAB}$ and $X^R = \texttt{BACCBA}$. Here $Z = \texttt{ACCB}$ is an LCS with $C = 4$, embedded in X at positions $i = (1, 3, 4, 6)$ and in X^R at the positions that correspond to $j = (5, 4, 3, 2)$ in X, so $k = 2$. Reading the endpoints of the nested right-pointing arrows (above) from left to right gives the palindrome $\texttt{A C C A}$ of length $2k = 4$.
- *Right ($k < C/2$).* $X = \texttt{ABCAB}$ and $X^R = \texttt{BACBA}$. Here $Z = \texttt{BCA}$ is an LCS with $C = 3$, with $i = (2, 3, 4)$ and $j = (5, 3, 1)$, so $k = 1$. Only one arrow points right, and it gives just $\texttt{B B}$, of length $2 < C$. Instead we use the left-pointing arrows (below). The arrow for $z_2 = \texttt{C}$ starts and ends at position 3, so that character is used once, in the middle. Reading the endpoints from left to right gives $\texttt{A C A}$, of length $2(C - k) - 1 = 3$.



Each arrow goes from i_t to j_t. Above: the right-pointing arrows ($t \le k$). Below: the left-pointing arrows ($t > k$). Left: an example with $k \ge C/2$. Right: an example with $k < C/2$, where $z_2 = \texttt{C}$ has $i_2 = j_2 = 3$.

In general, the C arrows are split into k right-pointing and $C - k$ left-pointing arrows, so one of the two groups contains at least half of the arrows. Each nested group gives a palindrome whose characters are the two endpoints of each arrow, so its length is about twice the size of the group, and hence at least C.

- *Case $k \ge C/2$ (at least half the arrows point right).* The positions $i_1 < \cdots < i_k < j_k < \cdots < j_1$ give the subsequence $z_1 \cdots z_k\, z_k \cdots z_1$ of X, a palindrome of length $2k \ge C$.
- *Case $k < C/2$ (more than half the arrows point left).* The positions $j_C < \cdots < j_{k+1} \le i_{k+1} < \cdots < i_C$ give the subsequence $z_C \cdots z_{k+1}\, z_{k+1} \cdots z_C$, a palindrome of length $2(C - k)$. If $j_{k+1} = i_{k+1}$, this position is used only once, so the middle character z_{k+1} appears once and the length is $2(C - k) - 1$. Since $C - k > C/2$ and both are integers, $C - k \ge (C + 1)/2$, so either way the length is at least C.

In both cases X has a palindromic subsequence of length at least C, so $L \ge C$.

Combining both directions, $L = C$. Computing X^R takes $O(n)$ time and the LCS table for two strings of length n takes $O(n^2)$ time, so the longest palindromic subsequence can be found in $O(n^2)$ time.

Remark. The proof shows the two *lengths* are equal. An arbitrary LCS of X and X^R need not itself be a palindrome: in the right example of the figure, `BCA` is an LCS of `ABCAB` and `BACBA`, and the proof turns it into the palindrome `ACA`.

(c) For $0 \le i \le m$ and $0 \le j \le n$,

$$\mathrm{S}(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ \mathrm{S}(i-1, j-1) + 1 & \text{if } x_i = y_j, \\ 0 & \text{if } x_i \neq y_j. \end{cases}$$

Explanation. If $i = 0$ or $j = 0$, one of the prefixes is empty, so the only common substring is the empty string. Otherwise, a nonempty common substring ending exactly at x_i and y_j must have x_i and y_j as its last character.

- If $x_i \neq y_j$, no nonempty common substring ends at both positions, so $\mathrm{S}(i,j) = 0$.
- If $x_i = y_j$, a common substring ending at x_i and y_j is a common substring ending exactly at x_{i-1} and y_{j-1} (possibly empty), followed by this character. Contiguity forces the earlier part to end exactly at x_{i-1} and y_{j-1}, and the longest such part has length $\mathrm{S}(i-1, j-1)$. So $\mathrm{S}(i,j) = \mathrm{S}(i-1, j-1) + 1$.

Note the difference from LCS: on a mismatch, LCS may skip a character and keep the best value so far, but a substring cannot skip characters, so the value resets to 0.

Final answer. Every common substring ends at some pair of positions (i,j), so the length of the longest common substring is

$$\max_{1 \le i \le m,\, 1 \le j \le n} \mathrm{S}(i,j),$$

Running time. The table has $(m+1)(n+1)$ entries. Filling it in row by row, each entry takes $O(1)$ time, since it depends only on the entry diagonally above it. We can track the maximum and its position while filling the table, so the total running time is $O(mn)$.

Solution 3:

(a) Let $\textsc{Profit}(l, r)$ return the maximum profit using only days $l, \ldots, r$, i.e. the maximum of $p_j - p_i$ over $l \le i \le j \le r$. The answer is $\textsc{Profit}(1, n)$.

```
1  def profit(p, l, r):
2      if l == r:
3          return 0                          # buy and sell on the same day
4      mid = (l + r) // 2
5      left  = profit(p, l, mid)             # buy and sell in the left half
6      right = profit(p, mid + 1, r)         # buy and sell in the right half
7      cross = max(p[mid+1 : r+1]) - min(p[l : mid+1])   # buy left, sell right
8      return max(left, right, cross)
```

Correctness. Split the days in half. The best trade either stays in the left half, stays in the right half, or buys in the left half and sells in the right half. The first two are handled by the recursive calls. For the third, any left day comes before any right day, so we simply buy at the lowest price on the left and sell at the highest price on the right.

For example, $8, 3, 6, 2 \mid 5, 9, 4, 7$ gives $\max\{\underbrace{6-3}_{\text{left}}, \underbrace{9-5}_{\text{right}}, \underbrace{9-2}_{\text{cross}}\} = 7$.

Running time. $T(n) = 2T(n/2) + O(n)$, the same recurrence as merge sort, so $T(n) = O(n \log n)$.

(b) The $O(n)$ term in part (a) comes from scanning each half for its minimum and maximum. Instead, let each call return three values for days $l, \ldots, r$: the best profit, the minimum price, and the maximum price.

```
1  def profit(p, l, r):           # returns (best, lo, hi)
2      if l == r:
3          return (0, p[l], p[l])
4      mid = (l + r) // 2
5      best_l, lo_l, hi_l = profit(p, l, mid)
6      best_r, lo_r, hi_r = profit(p, mid + 1, r)
7      best = max(best_l, best_r, hi_r - lo_l)
8      return (best, min(lo_l, lo_r), max(hi_l, hi_r))
```

The best profit is computed exactly as in part (a), since `hi_r` $-$ `lo_l` is the crossing profit. The minimum (maximum) price of the whole range is the smaller (larger) of the minimum (maximum) prices of the two halves, so the returned minimum and maximum are also correct.

Running time. Now each call does $O(1)$ work besides the recursive calls, so

$$T(n) = 2T(n/2) + O(1), \qquad T(1) = O(1).$$

The recursion tree has n leaves and $n-1$ internal nodes, each doing $O(1)$ work, so $T(n) = O(n)$.

(c) Let $\mathrm{OPT}(j)$ be the maximum profit using only days $1, \ldots, j$, and let $m_j = \min\{p_1, \ldots, p_j\}$ be the lowest price up to day j. Then

$$\mathrm{OPT}(1) = 0, \qquad \mathrm{OPT}(j) = \max\big\{\, \mathrm{OPT}(j-1),\ p_j - m_j \,\big\} \quad \text{for } j \geq 2,$$

and $m_1 = p_1$, $m_j = \min\{m_{j-1}, p_j\}$. The answer is $\mathrm{OPT}(n)$.

Explanation. Consider the best trade using days $1, \ldots, j$.

- If it sells before day j, it only uses days $1, \ldots, j-1$, so its profit is $\mathrm{OPT}(j-1)$.
- If it sells on day j, the profit is $p_j - p_i$ for some $i \leq j$, which is largest when p_i is the lowest price among days $1, \ldots, j$. So its profit is $p_j - m_j$.

$\mathrm{OPT}(j)$ is the larger of the two. For the example prices:

j	1	2	3	4	5	6	7	8
p_j	8	3	6	2	5	9	4	7
m_j	8	3	3	2	2	2	2	2
$p_j - m_j$	0	0	3	0	3	7	2	5
$\mathrm{OPT}(j)$	0	0	3	3	3	7	7	7

so the maximum profit is $\mathrm{OPT}(8) = 7$.

Running time. We compute m_j and $\mathrm{OPT}(j)$ for $j = 1, \ldots, n$ in order, each in $O(1)$ time from the previous values, so the total time is $O(n)$.

Solution 4:

(a)

$$\mathrm{OPT}(v) = \max_{(u,v)\in E} \big(\mathrm{OPT}(u) + w(u,v)\big),$$

where the inner maximum is taken to be 0 if v has no incoming edges. The length of the longest path in G is $\max_v \mathrm{OPT}(v)$.

Explanation. A longest path ending at v is either just v (length 0), or ends with some edge (u, v). In the second case, the part before v is a path ending at u, and it must be a longest one: otherwise we could swap in a longer path ending at u and get a longer path to v. So this case gives $\mathrm{OPT}(u) + w(u, v)$, and we try every incoming edge.

Order. $\mathrm{OPT}(v)$ depends on $\mathrm{OPT}(u)$ for every edge (u, v), so we compute the values in a topological order, which puts every u before v. Topological sorting takes $O(|V| + |E|)$ time, and the recurrence looks at each edge once, so the total running time is $O(|V| + |E|)$.

The example. One topological order is a, d, b, f, c, e, g. (Others work too, e.g. a, b, d, f, c, e, g.)

v	candidates $\mathrm{OPT}(u) + w(u, v)$	$\mathrm{OPT}(v)$	from
a	(no incoming edges)	0	—
d	a: $0 + 3 = 3$	3	a
b	a: $0 + 2 = 2$	2	a
f	d: $3 + 4 = 7$, b: $2 + 6 = 8$	8	b
c	a: $0 + 4 = 4$, d: $3 + 2 = 5$, b: $2 + 1 = 3$	5	d
e	f: $8 + 3 = 11$, c: $5 + 5 = 10$	11	f
g	f: $8 + 1 = 9$, c: $5 + 4 = 9$, e: $11 + 2 = 13$	13	e

(b) Think of the LCS table as a grid, where vertex (i, j) stands for the prefixes $x_1 \cdots x_i$ and $y_1 \cdots y_j$. Add the edges

- $(i-1, j) \to (i, j)$ with weight 0 (skip x_i),
- $(i, j-1) \to (i, j)$ with weight 0 (skip y_j),
- $(i-1, j-1) \to (i, j)$ with weight 1, only if $x_i = y_j$ (match x_i with y_j).

Acyclic. Every edge increases $i + j$, so no path can return to a vertex it has already visited.

Longest path = LCS. A path from $(0, 0)$ to (m, n) moves right and down through the grid, and each weight-1 edge it uses matches a character of X with an equal character of Y. Since i and

j never decrease, these matches appear in the same order in both strings, so they spell out a common subsequence, and the length of the path is the number of matches. Conversely, any common subsequence can be traced out this way: walk with weight-0 edges to each matched pair and take the diagonal edge there. So the longest path from $(0, 0)$ to (m, n) has length equal to the LCS.

Same recurrence. Applying part (a) to vertex (i, j) gives

$$\mathrm{OPT}(i,j) = \max\big\{\, \mathrm{OPT}(i-1,j),\ \ \mathrm{OPT}(i,j-1),\ \ \mathrm{OPT}(i-1,j-1)+1 \text{ (only if } x_i = y_j) \,\big\},$$

with $\mathrm{OPT}(i, j) = 0$ when $i = 0$ or $j = 0$, since those vertices only have weight-0 edges coming in. When $x_i \neq y_j$, this is exactly the LCS recurrence from class. When $x_i = y_j$, the class recurrence uses only $\mathrm{OPT}(i-1, j-1)+1$; the other two terms do not change the maximum, since adding one character to a string increases the LCS by at most 1, so $\mathrm{OPT}(i-1, j)$ and $\mathrm{OPT}(i, j-1)$ are at most $\mathrm{OPT}(i-1, j-1)+1$.