CSE 201
Homework Assignment 6
Solutions

1. **Moving on a checkerboard** (This is problem 15-6 on page 368 of the 2nd edition of CLRS.) Suppose that you are given an $n \times n$ checkerboard and a single checker. You must move the checker from the bottom (1^{st}) row of the board to the top (n^{th}) row of the board according to the following rule. At each step you may move the checker to one of three squares:

   - the square immediately above,
   - the square one up and one to the left (unless the checker is already in the leftmost column),
   - the square one up and one right (unless the checker is already in the rightmost column).

   Each time you move from square *x* to square *y*, you receive $p(x, y)$ dollars. The values $p(x, y)$ are known for all pairs (x, y) for which a move from *x* to *y* is legal. Note that $p(x, y)$ may be negative for some (x, y).

   Give an algorithm that determines a set of moves starting at the bottom row, and ending at the top row, and which gathers as many dollars as possible. Your algorithm is free to pick any square along the bottom row as a starting point, and any square along the top row as a destination in order to maximize the amount of money collected. Determine the runtime of your algorithm.

   Solution:
   Define $C[i, j]$ to be the maximum amount of money that can be collected in this process by moving a checker from any square on row 1, to the square at row i , column j. Obviously $C[1, j] = 0$ for all j in the range $1 \leq j \leq n$, since at least one move must be made to collect any money. Once the table entry $C[i, j]$ is known for all i and j ($1 \leq i \leq n, 1 \leq j \leq n$), the maximum amount of money that can be collected by moving from row 1 to row n is computed as $\max_{1 \leq j \leq n} C[n, j]$.

   Observe that if one knows an optimal sequence of moves leading to square $y = (i, j)$, where $i > 1$, then the last move in that sequence must originate in one of the three neighboring squares in row $i - 1$. These three squares have coordinates $(i - 1, j - 1)$ (if $j > 1$), $(i - 1, j)$ and $(i - 1, j + 1)$ (if $j < n$). Let that preceding square be denoted x.

   Claim: The subsequence of moves ending at *x* is itself an optimal sequence from row 1 to x.
   Proof: Suppose there exists a more valuable sequence from row 1 to square x. Then by following that sequence with a single move from x to y, we obtain a more valuable sequence from row 1 to square y than our original optimal one, a contradiction. Therefore any optimal sequence ending at $y = (i, j)$ consists of an optimal sequence to the predecessor x of y, followed by a single move from x to y. ■
   This problem therefore exhibits the required optimal substructure for a dynamic programming solution. Using the same notation as above, it is evident that $C[i, j] = C[y] = C[x] + p(x, y)$. Since the predecessor x is not known in advance, we have

$$C[i, j] = C[y] = \max_{x}(C[x] + p(x, y)),$$

   where the maximum is taken over all (at most 3) possible predecessors x, of y. It is now a simple matter to write an iterative algorithm to fill in the table C. Since we also wish to print out an optimal

sequence of moves, it is worthwhile to keep track of the predecessors as we fill in the table. Define $P[i,j]$ to be the predecessor of square (i,j) along an optimal sequence of moves starting in row 1, and ending at square (i,j), for $2 \le i \le n$ and $1 \le j \le n$.

OptimalSequence(p,n)
1. $C[1;\ 1 \cdots n] = (0, \ldots, 0)$ // the first row is initialized to all zeros
2. for $i = 2$ to n
3. for $j = 1$ to n
4. $y = (i,j)$
5. $x_0 = (i-1,j)$
6. $x_{-1} = \begin{cases} x_0 & \text{if } j = 1 \\ (i-1,j-1) & \text{if } j > 1 \end{cases}$
7. $x_1 = \begin{cases} x_0 & \text{if } j = n \\ (i-1,j+1) & \text{if } j < n \end{cases}$
8. $C[y] = C[x_{-1}] + p(x_{-1}, y)$
9. $P[y] = x_{-1}$
10. if $C[x_0] + p(x_0, y) > C[y]$
11. $C[y] = C[x_0] + p(x_0, y)$
12. $P[y] = x_0$
13. if $C[x_1] + p(x_1, y) > C[y]$
14. $C[y] = C[x_1] + p(x_1, y)$
15. $P[y] = x_1$
16. $k = 1$
17. for $j = 2$ to n
18. if $C[n,j] > C[n,k]$
19. $k = j$
20. return $(C[n,k], k, P)$

Lines 4-7 initialize y and its three possible predecessors: x_{-1}, x_0, x_1, which reduce to two when either $j = 1$ or $j = n$. Lines 8-15 determine the larger of $C[x_{-1}] + p(x_{-1}, y)$, $C[x_0] + p(x_0, y)$ and $C[x_1] + p(x_1, y)$, then set $C[y]$ and $P[y]$ accordingly. Lines 16-19 determine the maximum value in the n^{th} row of C, which is the value of an optimal sequence from row 1 to row n. That value, the column k where it is found, and the table of predecessors P are returned on line 20.

PrintSequence$(P, (i,j))$ (pre: P was returned by OptimalSequence())
1. if $i \ge 2$
2. PrintSequence$(P, P[i,j])$
3. print("move to square ", (i,j))
4. else
5. print("start at square ", (i,j))

PrintSequence$(P, (i,j))$ prints an optimal sequence starting at row 1 and ending at square (i,j). To print an optimal sequence ending at row n, call PrintSequence$(P, (n,k))$ where P, n and k were returned by OptimalSequence(). OptimalSequence() clearly runs in time $\Theta(n^2)$. The cost of PrintSequence() is the depth of the recursion, which is simply the number of print statements executed, i.e. $\Theta(n)$. ■

2. (This is Problem 15.2-1 on page 378 of CLRS.) Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $(5, 10, 3, 12, 5, 50, 6)$. Show the complete tables $m[i, j]$ and $s[i, j]$ described in the handout on Dynamic Programming.

Solution:
In this case $n = 6$ and $p[0 \cdots 6] = (5, 10, 3, 12, 5, 50, 6)$. Recall that

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i < j \end{cases}$$

for $1 \le i \le j \le n$. Also $s[i, j]$ is defined to be the value of k at which the minimum occurs in cell $m[i, j]$, for $1 \le i < j \le n$. With some effort, we have

$m[\ ]$		j 1	2	3	4	5	6
i	1	**0**	**150**	**330**	**405**	**1655**	**2010**
	2	-	**0**	**360**	**330**	**2430**	**1950**
	3	-	-	**0**	**180**	**930**	**1770**
	4	-	-	-	**0**	**3000**	**1860**
	5	-	-	-	-	**0**	**1500**
	6	-	-	-	-	**-**	**0**

and

$s[\ ]$		j 1	2	3	4	5	6
i	1	*	**1**	**2**	**2**	**4**	**2**
	2	-	*	**2**	**2**	**2**	**2**
	3	-	-	*	**3**	**4**	**4**
	4	-	-	-	*	**4**	**4**
	5	-	-	-	-	*	**5**
	6	-	-	-	-	**-**	*

The second table gives us the optimal parenthesization as

$$(A_1 \cdot A_2) \cdot \big((A_3 \cdot A_4) \cdot (A_5 \cdot A_6) \big)$$

■

3. (Read the **Rod-Cutting Problem** in section 15.1 pp. 360-369 of CLRS 3rd edition. This is problem 15.1-2 on p. 370.) Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the *density* of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

Counter Example

Let $n = 3$ and $p = (p_1, p_2, p_3) = (1, 10, 12)$. Then the densities p_i/i are $(1, 5, 4)$. The greedy strategy first cuts a rod of length 2 having price 10, leaving a rod of length 1 having price 1. The total price for this sequence of cuts is then $10 + 1 = 11$. This sequence is not optimal however, since the whole rod of length 3 (with no cuts) is worth 12. Therefore the greedy strategy does not produce an optimal sequence of rod cuts in this instance. ■