Problem 9. Paren Puzzle [15 points]

Your local school newspaper, *The TEX*, has started publishing puzzles of the following form:

Parenthesize $6 + 0 \cdot 6$ to maximize the outcome.	Parenthesize $0.1 \cdot 0.1 + 0.1$ to maximize the outcome.

Wrong answer: $6 + (0 \cdot 6) = 6 + 0 = 6.$ *Wrong answer:* $0.1 \cdot (0.1 + 0.1) = 0.1 \cdot 0.2 = 0.02.$
Right answer: $(6 + 0) \cdot 6 = 6 \cdot 6 = 36.$ *Right answer:* $(0.1 \cdot 0.1) + 0.1 = 0.01 + 0.1 = 0.11.$

To save yourself from tedium, but still impress your friends, you decide to implement an algorithm to solve these puzzles. The input to your algorithm is a sequence $x_0, o_0, x_1, o_1, \ldots, x_{n-1}, o_{n-1}, x_n$ of $n+1$ real numbers $x_0, x_1, \ldots, x_n$ and n operators $o_0, o_1, \ldots, o_{n-1}$. Each operator o_i is either addition ($+$) or multiplication ($\cdot$). Give a polynomial-time dynamic program for finding the optimal (maximum-outcome) parenthesization of the given expression, and analyze the running time.

Input: $\langle x_0\ o_0\ x_1\ o_1 \ldots o_{n-1}\ x_n \rangle$

其中 $x_0, \ldots, x_n$ 為 $n+1$ 个 real number

$o_0, \ldots, o_{n-1}$ 為 n 个 operation 會為乘, 加

Output: 可以得到最大值的 parenthesization 和其值

DP解: 定義子問題 $d[i,j]$ 為 $\langle x_i\ o_i\ x_{i+1}\ o_{i+1} \cdots x_j \rangle$ 下 可以得到的 parenthesization 最大值

case :
① $(x_i)\ o_i\ (x_{i+1} \ldots x_j) \Rightarrow d[i,i]\ o_i\ d[i+1, \ldots, j]$
② $(x_i\ o_i\ x_{i+1})\ o_{i+1}\ (x_{i+2}, \ldots, x_j) \Rightarrow d[i, i+1]\ o_{i+1}\ d[i+2, \ldots, j]$
$\vdots \qquad \vdots$
③ $(x_i\ o_i\ x_{i+1} \ldots\ x_{j-1})\ o_j\ (x_j) \Rightarrow d[i, j-1]\ o_{j-1}\ d[j-1, j]$

$$\therefore d[i,j] = \begin{cases} x_i & \text{if } i = j \\ 0 & \text{if } i > j \\ \max\limits_{i \le k < j} \{ d[i,k]\ o_k\ d[k,j] \} & \end{cases}$$

$\Rightarrow T(n) = O(n^2) \cdot O(n) = O(n^3)$ #.

Solution: The following dynamic program is the intended "correct" answer, though it ignores a subtle issue detailed below (which only three students identified, and received bonus points for). It is similar to the matrix-multiplication parenthesization dynamic program we saw in lecture, but with a different recurrence.

1. For subproblems, we use substrings $x_i, o_i, \ldots, o_{j-1}, x_j$, for each $0 \le i \le j \le n$. Thus there are $\Theta(n^2)$ subproblems.

2. To solve $DP[i, j]$, we guess which operation o_k is outermost, where $i \le k < j$. There are $j - i = O(n)$ choices for this guess.

3. The resulting recurrence relation is

$$\boxed{DP[i,j] = \max_{k=i}^{j-1} \Big(DP[i,k] \; o_k \; DP[k+1,j] \Big).}$$

The base cases are

$$DP[i,i] = x_i.$$

The running time per subproblem is $O(n)$.

4. The dynamic program uses either recursion plus memoization, or bottom-up table construction. A suitable acyclic order is by increasing length ℓ of substring, i.e.,

for $\ell = 0, 1, \ldots, n$:
  for $i = 0, 1, \ldots, n - \ell$:
    $j = i + \ell$

5. The value of the original problem is given by $DP[0, n]$. To actually reconstruct the parenthesization, we can remember and follow parent pointers (the argmax in addition to each $\max$). The overall running time is

$$\Theta(n^2) \cdot O(n) = O(n^3).$$