Computer Science C73 | October 19, 2022
Scarborough Campus | University of Toronto

Solutions for Homework Assignment #5

Answer to Question 1.

<u>*Definition of the subproblems to solve:*</u> For all j, $1 \le j \le n$, define

$$\begin{aligned} M[j] &= \max\{A[i] \times A[i+1] \times \ldots \times A[j] : \ 1 \le i \le j\} \\ m[j] &= \min\{A[i] \times A[i+1] \times \ldots \times A[j] : \ 1 \le i \le j\} \end{aligned} \tag{*}$$

(i.e., $M[j]$ and $m[j]$ are, respectively, the maximum and minimum product of the elements in subarrays of A that end in position j — where "the product of the elements" in a subarray of length 1 is taken to be just that element).

<u>*Recursive formula to compute each subproblem:*</u> The following are recursive formulas to compute $M[j]$ and $m[j]$ for all $j \in [1..n]$.

$$\begin{aligned} M[j] &= \begin{cases} A[1], & \text{if } j = 1 \\ \max(A[j], M[j-1] \cdot A[j], m[j-1] \cdot A[j]), & \text{if } j > 1 \end{cases} \\ m[j] &= \begin{cases} A[1], & \text{if } j = 1 \\ \min(A[j], M[j-1] \cdot A[j], m[j-1] \cdot A[j]), & \text{if } j > 1 \end{cases} \end{aligned} \tag{†}$$

<u>*Justification why (†) is a correct formula to compute (∗):*</u> To prove this, we consider three cases, depending on whether $A[j]$ is zero, positive, or negative.

Case 1. $A[j] = 0$. Then, the product of numbers in $A[i..j]$ is 0, for all i such that $1 \le i \le j$, and clearly the recursive formula also evaluates to 0, since it is the maximum of three terms, all of which are 0.

Case 2. $A[j] > 0$. In this case, to get the maximum product of numbers in $A[i..j]$, over all i such that $1 \le i \le j$, we want to multiply $A[j]$ by as ***big*** a quantity as possible; so the maximum product of numbers in $A[i..j]$ is the product of $A[j]$ times the ***maximum*** product of numbers in $A[i..j-1]$, over all i such that $1 \le i \le j-1$, i.e., $M[j-1]$, or $A[j]$, whichever is larger. Since $M[j-1] \ge m[j-1]$ (by definition) and $A[j] > 0$ (by the case hypothesis), this is also equal to $\max(A[j], M[j-1] \cdot A[j], m[j-1] \cdot A[j])$. So, $M[j]$ is computed correctly by the recursive formula in this case. A similar argument shows that $m[j]$ is computed correctly by the recursive formula in this case.

Case 3. $A[j] < 0$. In this case, to get the maximum product of numbers in $A[i..j]$, over all i such that $1 \le i \le j$, we want to multiply $A[j]$ by as ***small*** a quantity as possible, so the maximum product of numbers in $A[i..j]$ is the product of $A[j]$ times the ***minimum*** product of numbers in $A[i..j-1]$, over all i such that $1 \le i \le j-1$, i.e., $m[j-1]$, or $A[j]$, whichever is larger. Since $M[j-1] \ge m[j-1]$ (by definition) and $A[j] < 0$ (by the case hypothesis), this is also equal to $\max(A[j], M[j-1] \cdot A[j], m[j-1] \cdot A[j])$. So, $M[j]$ is computed correctly by the recursive formula in this case. A similar argument shows that $m[j]$ is computed correctly by the recursive formula in this case.

<u>*Solving the original problem:*</u> The quantity we are interested in (the maximum product of all elements of a subarray of A) is, by definition, simply $\max\{M[i] : i \in [1..n]\}$.

<u>*Pseudocode:*</u> Our dynamic programming algorithm computes $M[1], M[2], \ldots, M[n]$, in this order, using the above recursive formula, and returns the maximum of these values. Expressed in pseudocode, the algorithm is as follows (p keeps track of the maximum product of subarrays of A ending in positions examined so far):

```
M[1] := m[i] := A[1]
p := M[1]
for i := 2 to n do
    M[i] := max(A[i], M[i−1]·A[i], m[i−1]·A[i])
    m[i] := min(A[i], M[i−1]·A[i], m[i−1]·A[i])
    p := max(p, M[i])
return p
```

Running time: The running time is dominated by the for loop, which obviously takes $\Theta(n)$ time.

Answer to Question 2.

Definition of the subproblems to solve: Let $S[1..n]$ be the given string of symbols over $\{a, b, c\}$. For all $i, j \in [1..n]$ such that $i \leq j$, and all $x \in \{a, b, c\}$, define

$$P[i,j,x] = \begin{cases} \textbf{true}, & \text{if } S[i..j] \text{ can be parenthesized s.t. the value of the resulting expression is } x, \\ \textbf{false}, & \text{otherwise} \end{cases} \quad (*)$$

Recursive formula to compute each subproblem:

$$P[i,j,x] = \begin{cases} \textbf{true}, & \text{if } i = j \text{ and } S[i] = x \\ \textbf{false}, & \text{if } i = j \text{ and } S[i] \neq x \\ \textbf{true}, & \text{if } i < j \text{ and } \exists k \in [i..j-1] \text{ and } y, z \in \{a, b, c\} \text{ such that} \\ & \qquad P[i,k,y] = \textbf{true},\ P[k+1,j,z] = \textbf{true} \text{ and } yz = x \\ \textbf{false}, & \text{otherwise} \end{cases} \quad (\dagger)$$

Justification why (†) is a correct formula to compute (∗): The base cases (when $i = j$) are obvious. For the induction step (when $i < j$) note that the string $S[i..j]$ can be parenthesized so that the resulting expression evaluates to z if the string is the concatenation of two smaller strings $S[i..k]$ and $S[k+1..j]$ that can be parenthesized to evaluate to two values whose "product" is x.

Solving the original problem: By the definition (*), we wish to determine the truth value of $P[1, n, a]$.

Pseudocode: The algorithm is shown in pseudocode below. We assume that M is the "multiplication table" of the given operation; that is, for any $x, y \in \{a, b, c\}$, $M[x, y]$ gives the value of the "product" xy.

```
ParenValue(S[1..n])
for i := 1 to n do
    for x ∈ {a, b, c} do
        if M[S[i], S[i]] = x then P[i, i, x] = true
        else P[i, i, x] = false
for d := 1 to n − 1 do
    for i := 1 to n − d do
        j := i + d
        for x ∈ {a, b, c} do P[i, j, x] := false
        for k := i to j − 1 do
            for y ∈ {a, b, c} do
                for z ∈ {a, b, c} do
                    if P[i, k, y] = true and P[k + 1, j, z] = true then P[i, j, M[y, z]] = true
return P[1, n, a]
```

Running time: The running time of this algorithm is $O(n^3)$, as it is dominated by the triply-nested loops "**for** d", "**for** i", "**for** k", each of which is executed at most n times. (Note that the "**for** x", "**for** y", and "**for** z" loops are executed a constant number of times each.)

Answer to Question 3.

<u>*Definition of the subproblems to solve:*</u> For all $i \in [1..m], j \in [1..n]$, define

$$MCS[i,j] = \text{minimum cost of a seam of } P[1..i, 1..n] \text{ \underline{that ends in pixel (i,j)}} \qquad (*)$$

(Note that we are restricting the image to the first i rows, and we minimize over the seams that end in a particular pixel of the last row or this restricted image.)

<u>*Recursive formula to compute each subproblem:*</u>

$$MCS[i,j] = \begin{cases} C(1,j), & \text{if } i = 1 \\ \infty, & \text{if } i \neq 1, \text{ and } j = 1 \text{ or } j = n \\ \min\big(MCS[i-1,j-1], MCS[i-1,j], MCS[i-1,j+1]\big) + C(i,j), & \text{otherwise} \end{cases} \qquad (\dagger)$$

<u>*Justification why (†) is a correct formula to compute (∗):*</u> The base cases (when $i = 1$) is obvious, since there is a single seam of a 1-row image that ends in a particular pixel, namely the pixel itself. The cases $j = 1$ and $j = n$ are also obvious since a seam that contains a pixel from the first or last column has infinite cost. For the remaining case note that a seam that ends in pixel (i,j) consists of a seam that ends in one of pixels $(i-1,j-1)$, $(i-1,j)$, or $(i-1,j+1)$, followed by the pixel (i,j). A straightforward cut-and-paste argument shows that the the portion of the seam that ends in one of pixels $(i-1,j-1)$, $(i-1,j)$, or $(i-1,j+1)$, must be of minimum cost among the seams that end there.

<u>*Solving the original problem:*</u> By the definition (*), we wish to determine a seam of $P[1..m, 1..n]$ whose cost is $\min\{MCS[m,j] :\ 1 \leq j \leq n\}$. Having computed $MCS(i,j)$ for all values of i and j based on (†) we can then find the j that minimizes the last row of this array, and work backwards to find the pixels of the wanted seam.

<u>*Pseudocode:*</u> The algorithm is shown in pseudocode below.

MinCostSeam($P[1..m, 1..n]$)
▶ compute $C(i,j)$ the cost of every pixel
1 **for** $i := 1$ **to** m **do** $C(i,1) := C(i,n) := \infty$
2 **for** $i := 1$ **to** m **do**
3   **for** $j := 2$ **to** $n-1$ **do**
4         $C(i,j) := |P[i,j] - P[i,j-1]| + |P[i,j] - P[i,j+1]|$
▶ compute the min cost $MCS[i,j]$ of a seam ending at (i,j)
5 **for** $j := 1$ **to** n **do** $MCS[1,j] := C[1,j]$
6 **for** $i := 1$ **to** m **do** $MCS[i,1] := MCS[i,n] := \infty$
7 **for** $i := 2$ **to** $m-1$ **do**
8   **for** $j := 2$ **to** $n-1$ **do**
9         $MCS[i,j] := \min\big(MCS[i-1,j-1], MCS[i-1,j], MCS[i-1,j+1]\big) + C(i,j)$
▶ compute the min cost seam S
10 $minj := \arg\min_{1 \leq j \leq n} MCS[m,j]$ ▶ $minj$ is a j that minimizies $MCS[m,j]$
11 $S := (m, minj)$ ▶ $(m, minj)$ is the last pixel of a minimum cost seam
12 **for** $i := m-1$ **downto** 1 **do** ▶ find the row i pixel of the minimum cost seam and add it to S
13   **if** $MCS[i+1, minj] = MCS[i, minj-1] + C(i, minj)$ **then** $minj := minj - 1$
14   **elsif** $MCS[i+1, minj] = MCS[i, minj+1] + C(i, minj)$ **then** $minj := minj + 1$
15   prepend $(i, minj)$ to S
16 **return** S

<u>*Running time:*</u> The running time of this algorithm is $O(mn)$, as it is dominated by the doubly-nested loops in lines 2-4 and 7-9.