CSCI-4470 HW-3a

Due: Oct 3 2023, Tuesday

Homework 3

Jun Wang
ID: 811574679

Upload a soft copy of your answers **(single pdf file)** to the submission link on elc before the due date. The answers to the homework assignment should be your own individual work. Make sure to show all the work/steps in your answer to get full credit. Answers without steps or explanation will be given zero.

Extra credit: There is 5 percentage extra credit if you don't submit hand written homework including the figures. You can use latex or any other tool to write your homework. For figures you can use any drawing tool and include the figure as a jpeg or a png file in your latex file.

1. (20 points) Using Figure 14.5 as a model, Find the optimal way to place parenthesis for a chain of multiplications with dimension described by `〈9,3,8,2,5,6〉`. Provide both the cost table and the selection table in your answer.

$$m[i,j] = \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j\}$$

A_1	A_2	A_3	A_4	A_5
9×3	3×8	8×2	2×5	5×6
$d_0\ d_1$	$d_1\ d_2$	$d_2\ d_3$	$d_3\ d_4$	$d_4\ d_5$

$m[i,j]$	1	2	3	4	5
1	0	216	102	192	270
2	–	0	48	78	144
3	–	–	0	80	156
4	–	–	–	0	60
5	–	–	–	–	0

$k[i,j]$	1	2	3	4	5
1	0	1	1	3	3
2	–	0	2	3	3
3	–	–	0	3	3
4	–	–	–	0	4
5	–	–	–	–	0

$$m[1,2] = \min_{1 \le k < 2} \begin{cases} k = 1: & m[1,1] + m[2,2] + d_0 \times d_1 \times d_4 \\ k = 1: & 0 + 0 + 9 \times 3 \times 8 = 216 \end{cases}$$

$$m[2,3] = \min_{2 \le k < 3} \begin{cases} k = 2: & m[2,2] + m[3,3] + d_1 \times d_2 \times d_3 \\ k = 2: & 0 + 0 + 3 \times 8 \times 2 = 48 \end{cases}$$

$$m[3,4] = \min_{3 \le k < 4} \begin{cases} k = 3: & m[3,3] + m[4,4] + d_2 \times d_3 \times d_4 \\ k = 3: & 0 + 0 + 8 \times 2 \times 5 = 80 \end{cases}$$

$$m[4,5] = \min_{4 \le k < 5} \begin{cases} k = 4: & m[4,4] + m[5,5] + d_3 \times d_4 \times d_5 \\ k = 4: & 0 + 0 + 2 \times 5 \times 6 = 60 \end{cases}$$

$$m[1,3] = \min_{1 \le k < 3} \begin{cases} k = 1: & m[1,1] + m[2,3] + d_0 \times d_1 \times d_3 \\ k = 1: & 0 + 48 + 9 \times 3 \times 2 = 102 \\ k = 2: & m[1,2] + m[3,3] + d_0 \times d_2 \times d_3 \\ k = 2: & 216 + 0 + 9 \times 8 \times 2 = 360 \end{cases}$$

$$m[2,4] = \min_{2 \le k < 4} \begin{cases} k = 2: & m[2,2] + m[3,4] + d_1 \times d_2 \times d_4 \\ k = 2: & 0 + 80 + 3 \times 8 \times 5 = 200 \\ k = 3: & m[2,3] + m[4,4] + d_1 \times d_3 \times d_4 \\ k = 3: & 48 + 0 + 3 \times 2 \times 5 = 78 \end{cases}$$

$$m[3,5] = \min_{3 \le k < 5} \begin{cases} k = 3: & m[3,3] + m[4,5] + d_2 \times d_3 \times d_5 \\ k = 3: & 0 + 60 + 8 \times 2 \times 6 = 156 \\ k = 4: & m[3,4] + m[5,5] + d_2 \times d_4 \times d_5 \\ k = 4: & 80 + 0 + 8 \times 5 \times 6 = 320 \end{cases}$$

$$m[1,4] = \min_{1 \le k < 4} \begin{cases} k = 1: & m[1,1] + m[2,4] + d_0 \times d_1 \times d_4 \\ k = 1: & 0 + 78 + 9 \times 3 \times 6 = 240 \\ k = 2: & m[1,2] + m[3,4] + d_0 \times d_2 \times d_4 \\ k = 2: & 216 + 80 + 9 \times 8 \times 5 = 656 \\ k = 3: & m[1,3] + m[4,4] + d_0 \times d_3 \times d_4 \\ k = 3: & 102 + 0 + 9 \times 2 \times 5 = 192 \end{cases}$$

$$m[2,5] = \min_{2 \le k < 5} \begin{cases} k = 2: & m[2,2] + m[3,5] + d_1 \times d_2 \times d_5 \\ k = 2: & 0 + 156 + 3 \times 8 \times 6 = 300 \\ k = 3: & m[2,3] + m[4,5] + d_1 \times d_3 \times d_5 \\ k = 3: & 48 + 60 + 3 \times 2 \times 6 = 144 \\ k = 4: & m[2,4] + m[5,5] + d_1 \times d_4 \times d_5 \\ k = 4: & 78 + 0 + 3 \times 5 \times 6 = 168 \end{cases}$$

$$m[1,5] = \min_{1 \le k < 5} \begin{cases} k = 1: & m[1,1] + m[2,5] + d_0 \times d_1 \times d_5 \\ k = 1: & 0 + 144 + 9 \times 3 \times 6 = 306 \\ k = 2: & m[1,2] + m[3,5] + d_0 \times d_2 \times d_5 \\ k = 2: & 216 + 156 + 9 \times 8 \times 6 = 804 \\ k = 3: & m[1,3] + m[4,5] + d_0 \times d_3 \times d_5 \\ k = 3: & 102 + 60 + 9 \times 2 \times 6 = 270 \\ k = 4: & m[1,4] + m[5,5] + d_0 \times d_4 \times d_5 \\ k = 4: & 192 + 0 + 9 \times 5 \times 6 = 462 \end{cases}$$

$m[i,j]$	1	2	3	4	5	$k[i,j]$	1	2	3	4	5
1	0	216	102	192	270	1	0	1	1	3	3
2	–	0	48	78	144	2	–	0	2	3	3
3	–	–	0	80	156	3	–	–	0	3	3
4	–	–	–	0	60	4	–	–	–	0	4
5	–	–	–	–	0	5	–	–	–	–	0

Optimal Parenthesization:

Step 1: Multiply A_2 and A_3

1. $A_2(3 \times 8)$ and $A_3(8 \times 2)$
2. Cost: $3 \times 8 \times 2 = 48$

Step 2: Multiply A_1 with the result of Step 1

1. $A_1(9 \times 3)$ and result from Step 1 (3×2)
2. Cost: $9 \times 3 \times 2 = 54$

Step 3: Multiply A_4 and A_5

1. $A_4(2 \times 5)$ and $A_5(5 \times 6)$
2. Cost: $2 \times 5 \times 6 = 60$

Step 4: Multiply the results of Step 2 and Step 3

1. Result from Step 2 (9×2) and result from Step 3 (2×6)
2. Cost: $9 \times 2 \times 6 = 108$

Total Cost: $48 + 54 + 60 + 108 = 270$

This leads to the parenthesization $((A_1(A_2A_3))(A_4A_5))$.

> $((A1(A2A3))(A4A5))$

---

2. (20 points) Given an array `A` of `n` integers (could be positive, negative, or 0), find a subarray of `A` with maximum total product. For example, if `A = [6, -3, -10, 0, 2]` the maximum product is 180 – the product of `A[1 : 3]`, and if `A = [-1, 2, 3, 4, 5, 0]` the product is 120, the product of `A[2 : 5]`. Find a dynamic programming solution to this problem. Be sure to include a method for determining which subarray gives the maximum product.

To solve this question by using a dynamic programming approach that keeping track of the maximum and minimum product ending at each position in the array.

Notes: keeping track of the minimum product is that a smaller number can become larger when multiplied by a negative number. (`max_val` and `min_val`)

1. Initialization:

Initialize variables to keep track of the maximum and minimum product ending at the current position, and the start and end indices of the maximum product subarray.

> `max_val` and `min_val` keep track of the maximum and minimum product ending at the current position.

> `start` and `end` keep track of the start and end indices of the maximum product subarray.

> `temp_start` is used to keep track of the possible start index of a new subarray.

2. Iterate through the Array:

Iterate through the array. For each number, compute the new maximum and minimum product ending at that position.

> `temp_start` is updated whenever a new subarray could potentially start (when `max_val` is reset to the current element).

3. Update the Result and Indices:

Update the result and the start and end indices whenever a new maximum product is found.

> `start` and `end` are updated whenever a new maximum product is found.

Determine the Recurrence Relation:

The recurrence relation of the dynamic programming solution of finding the maximum product subarray.

In this question, Let `max_val[i]` and `min_val[i]` be the maximum and minimum product ending at position `i` in the array.

The recurrence relations are:

$$\text{max\_val}[i] = \max(A[i], A[i] \times \text{max\_val}[i-1], A[i] \times \text{min\_val}[i-1])$$

$$\text{min\_val}[i] = \min(A[i], A[i] \times \text{max\_val}[i-1], A[i] \times \text{min\_val}[i-1])$$

- $A[i]$ is the current element in the array.
- $\text{max\_val}[i-1]$ and $\text{min\_val}[i-1]$ are the maximum and minimum product ending at the previous position in the array.
- The dynamic programming aspect is handled by `max_val` and `min_val`, which keep track of the maximum and minimum product subarray ending at each position in the array as we iterate through it.
- The method for determining which subarray gives the maximum product is handled by keeping track of the `start` and `end` indices. Whenever a new maximum product is found (`max_val > max_product`), the `start` and `end` indices are updated.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution{
public:
    pair<long long, pair<int, int>> maxProduct(vector<int> A, int n) {
        long long min_val = A[0];
        long long max_val = A[0];
        long long max_product = A[0];
        int start = 0, end = 0, temp_start = 0;

        for (int i = 1; i < n; i++) {
            if (A[i] < 0) swap(max_val, min_val);

            max_val = max((long long)A[i], max_val * A[i]);
            min_val = min((long long)A[i], min_val * A[i]);

            if (max_val > max_product) {
                max_product = max_val;
                start = temp_start;
                end = i;
            }

            if (max_val == A[i]) temp_start = i;
        }

        return {max_product, {start, end}};
    }
};

int main () {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> A(n);
        for (int i = 0; i < n; i++) {
            cin >> A[i];
        }
        Solution ob;
        auto ans = ob.maxProduct(A, n);
        cout << "Maximum Product: " << ans.first << "\n";
        cout << "Subarray: [";
        for (int i = ans.second.first; i <= ans.second.second; i++) {
            cout << A[i];
            if (i < ans.second.second) cout << ", ";
        }
        cout << "]\n";
    }
    return 0;
}
```

3. (30 points) Consider the Edit Distance problem where edit operations have a cost and the goal is to maximize editing score in which matches are awarded positive scores. Consider editing between the two strings "Distance" and "Destiny". Assume a match is awarded score +4, an insertion or deletion is penalized with -2, and a mismatch with -4.

[a] Write the DP recursive solution (or equation) for the above problem.

Let $dp[i][j]$ be the maximum score for converting the first i characters of "Distance" to the first j characters of "Destiny". Let m, x, and y be the scores for match, mismatch, and gap, respectively. The recursive relation can be expressed as:

$$dp[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ dp[i-1][j-1] + m & \text{if characters match} \\ \max(dp[i-1][j] + x, dp[i][j-1] + y, dp[i-1][j-1] + x) & \text{otherwise} \end{cases}$$

[b] Using the recursive solution in part a create a dynamic programming table for the problem above using the two given strings.

Consider the strings "Distance" and "Destiny". The dynamic programming table is as follows:

```
       D  e  s  t  i  n  y
    0 -2 -4 -6 -8 -10 -12 -14
D -2  4  2  0 -2  -4  -6  -8
i -4  2  8  6  4   2   0  -2
s -6  0  6 12 10   8   6   4
t -8 -2  4 10 16  14  12  10
a-10 -4  2  8 14  20  18  16
n-12 -6  0  6 12  18  24  22
c-14 -8 -2  4 10  16  22  28
e-16-10 -4  2  8  14  20  26
```

[c] Looking at the table give the total cost of editing and sequence of operations performed to achieve that editing.

From the table, the total cost of editing is 26. The sequence of operations is:

1. Match D-D
2. Insert e
3. Match s-s
4. Match t-t
5. Match i-i
6. Match n-n
7. Substitute c-y
8. Match e-e