Lab03-Matroid & Dynamic Programming

CS2308-Algorithm and Complexity, Xiaofeng Gao, Spring 2022.

$*$ If there is any problem, please contact TA Hongjie Fang.

$*$ Name: Zhenran Xiao   Student ID: 520030910281   Email: xiaozhenran@sjtu.edu.cn

1. Let S be a finite set and let $S_1, S_2, \cdots, S_n$ be a partition of S into non-empty disjoint subsets. Define structure $(S, \mathcal{I})$ by the condition that

$$\mathcal{I} = \{I \mid I \subseteq S, |I \cap S_i| \le k_i \text{ for } i = 1, 2, \cdots, n\}$$

where $k_1, k_2, \cdots, k_n$ are positive integers. Does $(S, \mathcal{I})$ form a matroid? Why or why not? (*Hint: you may start from a simple case that $k_1 = k_2 = \cdots = k_n = 1$.*)

Solution.
$(S, \mathcal{I})$ form a matroid. Proof is following.

Hereditary:
If A $\subset$ B and B $\in \mathcal{I}$, meaning

$$B \subseteq S, |B \cap S_i| \le k_i \text{ for } i = 1, 2, \cdots, n$$

Obviously,

$$A \subseteq S, |A \cap S_i| \le k_i \text{ for } i = 1, 2, \cdots, n$$

Namely, A $\in \mathcal{I}$.

Exchange Property:
Assume that $k_1 = k_2 = \cdots = k_n = 1$, so we can easily know set I is composed of each one or zero element selected from $S_1, S_2, \cdots, S_n$. If A, B $\in \mathcal{I}$, |A|<|B|, there must exists an S_p that B has an element x selected from it but A doesn't have. Namely $x \in B \backslash A$. We create set $M = A \cup \{x\}$. Before, $|A \cap S_p| = 0$, now, $|M \cap S_p| = 1 \le k_p$. For other S_i, M is the same as A. So

$$M \subseteq S, |M \cap S_i| \le k_i \text{ for } i = 1, 2, \cdots, n$$

Namely, $A \cup \{x\} \in \mathcal{I}$.
If k_i are bigger, the proof is in a similar way. □

2. Given n matrices $A_1, A_2, \cdots, A_n$ and $(n+1)$ positive integers $m_1, m_2, \cdots, m_{n+1}$. The i-th matrix A_i $(1 \le i \le n)$ has the size of $m_i \times m_{i+1}$. Our goal is to calculate $A_1 \times A_2 \times \cdots \times A_n$. For any two matrix $P_{p\times q}$ and $Q_{q\times r}$, the computational cost of $P \times Q$ is $p \cdot q \cdot r$. Due to the combination law of matrix multiplication, we can arbitrarily add parentheses to change the calculation order without changing the final result. Since different methods of adding parentheses have different computational costs, please design an algorithm to minimize the total computational costs. You need to briefly describe your algorithm and write it in pseudo-code, then analyze the time complexity and space complexity of your algorithm.

Example. A_1 is a 4×2 matrix, A_2 is a 2×3 matrix, A_3 is a 3×1 matrix. If we calculate $A_1 \times A_2 \times A_3$ as the original order, the total computational costs are $4\times2\times3+4\times3\times1 = 36$. But if we calculate it as $A_1 \times (A_2 \times A_3)$, the total computational costs are $2\times3\times1+4\times2\times1 = 14$. In this example, your algorithm should output the minimum total computational cost 14.

Solution.
Let $c[i,j]$ be the minimum cost computing $A_iA_{i+1}\cdots Aj$, then we get the recursive formula

$$c[i,j] = \begin{cases} 0 & i = j \\ min\{c[i,k] + c[k+1,j] + m_{i-1}m_km_j\} & i < j \end{cases}$$

For every k from i to $j-1$, calculate the cost and find the minimum situation.
In the algorithm, $c[1,\cdots,n,1,\cdots,n]$ stores the cost $c[i,j]$; $s[1,\cdots,n,2,\cdots,n]$ records the segmentation point k corresponding to the optimal cost $c[i,j]$.

Algorithm 1: $minCost(array[m_1,...,m_{n+1}])$

```
Input: array[$m_1,...,m_{n+1}$]
Output: computational cost, an interger
1  Let $c[1,\cdots,n,1,\cdots,n]$ and $s[1,\cdots,n,2,\cdots,n]$ be new tables;
2  for $i \leftarrow 1$ to $n$ do
3      $c[i,i] = 0$;
4  for $l \leftarrow 2$ to $n$ do
5      for $i \leftarrow 1$ to $n$-$l$+1 do
6          $j = i + l - 1$;
7          $c[i,j] = \infty$;
8          for $k \leftarrow i$ to $j$-1 do
9              $q = c[i,k] + c[k+1,j] + m_{i-1}m_km_j$;
10             if $q < c[i,j]$ then
11                 $c[i,j] = q$;
12                 $s[i,j] = k$;
13 return $c[1,n]$;
```

The time complexity is

$$T(n) = n + (n-1)^3 \sim O(n^3)$$

The space complexity is

$$S(n) = n^2 + n(n-1) \sim O(n^2)$$

□

3. You are given an $m \times n$ matrix A with integer elements. Initially, you are located at the top-left corner $(1,1)$, and in each step, you can only move right or down in the matrix. Among all possible paths starting from the top-left corner $(1,1)$ and ending in the bottom-right corner (m,n), find the path with the maximum non-negative product. The product of a path is the product of all integers in the grid cells visited along the path.

   (a) If all elements in A are non-negative integers, design an algorithm to solve the problem. Your algorithm should return the maximum non-negative product as well as the optimal path. Write the algorithm in pseudo-code and analyze its time complexity.

   (b) If the elements in A are arbitrary integers, design an algorithm to solve the problem. Your algorithm should return the maximum non-negative product. Specially, if the maximum product is negative, your algorithm should return -1. Briefly describe your algorithm and analyze its time complexity.

Example. An example of the problem is illustrated in Fig. 1. The path marked in blue is the optimal path, whose product is $1 \times 1 \times (-2) \times (-4) \times 1 = 8$.

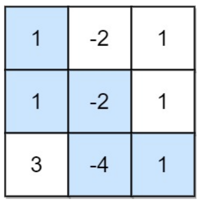

Figure 1: An Example of the Maximum Non-negative Product Path Problem

Solution.
(a)
Let $maxp[i,j]$ be the maximun product moving to $A[i,j]$, then we get the recursive formula

$$maxp[i,j] = \begin{cases} A[1,1] & i=1, j=1 \\ maxp[i,j-1] \times A[i,j] & i=1, j \neq 1 \\ maxp[i-1,j] \times A[i,j] & i \neq 1, j=1 \\ max(maxp[i,j-1], maxp[i-1,j]) \times A[i,j] & i \neq 1, j \neq 1 \end{cases}$$

In the algorithm, $maxp[1,\cdots,m,1,\cdots,n]$ stores the maximum product $maxp[i,j]$; $path[1,\cdots,m,1,\cdots$
records the path.

Algorithm 2: *optPath_nn*$(A_{m\times n})$

Input: an m× n matrix A with non-negative integer elements.
Output: the maximum non-negative product and the optimal path

```
Let $maxp[1,\cdots,m,1,\cdots,n]$ and $path[1,\cdots,m,1,\cdots,n]$ be new tables;
$maxp[1][1] \leftarrow A[1,1]$;
for $j \leftarrow 2$ to $n$ do
    $maxp[1][j] \leftarrow maxp[1][j-1] \times A[1,j]$;
for $i \leftarrow 2$ to $m$ do
    $maxp[i][1] \leftarrow maxp[i-1][1] \times A[j,1]$;
for $i \leftarrow 2$ to $m$ do
    for $j \leftarrow 2$ to $n$ do
        $maxp[i,j] \leftarrow max(maxp[i,j-1], maxp[i-1,j]) \times A[i,j]$;
        if $maxp[i,j-1] > maxp[i-1,j]$ then
            $path[i,j-1] \leftarrow$ "$\rightarrow$";
        else if $maxp[i,j-1] < maxp[i-1,j]$ then
            $path[i-1,j] \leftarrow$ "$\downarrow$";
        else
            $path[i,j-1] \leftarrow$ "$\rightarrow$";
            $path[i-1,j] \leftarrow$ "$\downarrow$";
return $maxp[m,n]$ and $path[1,\cdots,m,1,\cdots,n]$;
```

The time complexity is

$$T(n) = m + n + (m-1)(n-1) \sim O(mn)$$

The space complexity is

$$S(n) = mn + mn \sim O(mn)$$

(b)
Since there are positive and negative elements in the matrix, in order to obtain the maximum product, we shouldn't only store the maximum product in the process of moving. For example, if the current element is negative, the left is negative, the upper is positive, the upper is larger than the left, but obviously the left can make the product larger. Therefore, we need to store both the minimum and maximum of the product as it moves.

Algorithm 3: *optPath*($A_{m\times n}$)

```
Input: an m× n matrix A with integer elements.
Output: the maximum non-negative product and the optimal path
Let $maxp[1,\cdots,m,1,\cdots,n]$, $minp[1,\cdots,m,1,\cdots,n]$ and $path[1,\cdots,m,1,\cdots,n]$ be new tables;
$maxp[1][1] \leftarrow A[1,1]$;
$minp[1][1] \leftarrow A[1,1]$;
for $j\leftarrow$ 2 to $n$ do
    $maxp[1][j] \leftarrow maxp[1][j-1] \times A[1,j]$;
    $minp[1][j] \leftarrow minp[1][j-1] \times A[1,j]$;
for $i\leftarrow$ 2 to $m$ do
    $maxp[i][1] \leftarrow maxp[i-1][1] \times A[j,1]$;
    $minp[i][1] \leftarrow minp[i-1][1] \times A[j,1]$;
for $i\leftarrow$ 2 to $m$ do
    for $j\leftarrow$ 2 to $n$ do
        if $A[i][j] >= 0$ then
            $maxp[i,j] \leftarrow max(maxp[i,j-1], maxp[i-1,j]) \times A[i,j]$;
            $minp[i,j] \leftarrow min(minp[i,j-1], minp[i-1,j]) \times A[i,j]$;
            if $maxp[i,j-1] > maxp[i-1,j]$ then
                $path[i,j-1] \leftarrow$ "→";
            else if $maxp[i,j-1] < maxp[i-1,j]$ then
                $path[i-1,j] \leftarrow$ "↓";
            else
                $path[i,j-1] \leftarrow$ "→";
                $path[i-1,j] \leftarrow$ "↓";
        else
            $maxp[i,j] \leftarrow min(minp[i,j-1], minp[i-1,j]) \times A[i,j]$;
            $minp[i,j] \leftarrow max(maxp[i,j-1], maxp[i-1,j]) \times A[i,j]$;
            if $minp[i,j-1] < minp[i-1,j]$ then
                $path[i,j-1] \leftarrow$ "→";
            else if $minp[i,j-1] > minp[i-1,j]$ then
                $path[i-1,j] \leftarrow$ "↓";
            else
                $path[i,j-1] \leftarrow$ "→";
                $path[i-1,j] \leftarrow$ "↓";
if $maxp[m,n] < 0$ then
    return -1
else
    return $maxp[m,n]$ and $path[1,\cdots,m,1,\cdots,n]$;
```

The time complexity is

$$T(n) = m + n + (m-1)(n-1) \sim O(mn)$$

The space complexity is

$$S(n) = 2mn + mn \sim O(mn)$$

□

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.