> You should submit your code to caojiahao13@ruc.edu.cn BEFORE 11-2-7:30 pm

1. C++ exercise

1.1 Q4 in Exercise_II :

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example:

```
Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

1.2 Longest Common Subsequence

Given two strings `text1` and `text2`, return the length of their longest common subsequence.

A *subsequence* of a string is a new string generated from the original string with some characters(can be none) deleted without changing the relative order of the remaining characters. (eg, "ace" is a subsequence of "abcde" while "aec" is not). A *common subsequence* of two strings is a subsequence that is common to both strings.

If there is no common subsequence, return 0.

To avoid using too many loops, you may try to use dynamic programming.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

2. Matrix multiplication

Implement (a) and (b) using **Armadillo**.

Given two matrices $\mathbf{A}, \mathbf{B} \in R^{2^n \times 2^n}$ for a $n \in \{1, 2, 3, \dots\}$. We want to calculate the matrix product C as

$$\mathbf{C} = \mathbf{A}\mathbf{B} \quad \mathbf{C} \in R^{2^n \times 2^n}$$

(a) Use **Divide and Conquer Strategy** by partition A, B and C into equally sized block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}$$

Will this lead to a better time complexity? (考虑时间复杂度的阶数)

(b) The **Strassen algorithm** defines instead new matrices:

$$\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

Then we have:

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}$$

Consider why this reduce time complexity.

If time allows, you can check which algorithm is faster as n goes large (e.g. n=5,6,7,8):

(i)Direct calculation using three loops; (ii)algorithm in (a); (iii)algorithm in (b) (iv): using matrix product `*` in Armadillo.