3 栈和队列

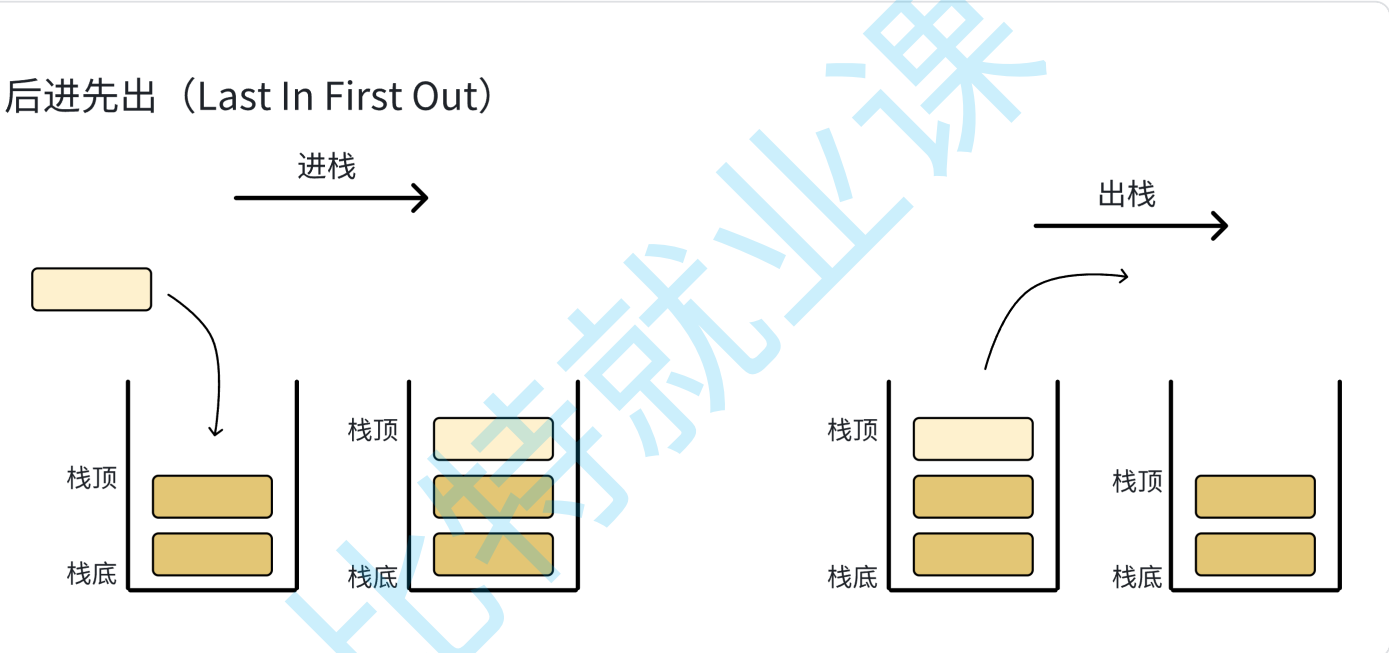
1. 栈

1.1 概念与结构

栈：一种特殊的线性表，其只允许在固定的一端进行插入和删除元素操作。进行数据插入和删除操作的一端称为栈顶，另一端称为栈底。栈中的数据元素遵守后进先出LIFO（Last In First Out）的原则。

压栈：栈的插入操作叫做进栈/压栈/入栈，入数据在栈顶。

出栈：栈的删除操作叫做出栈。**出数据也在栈顶**。



> 栈底层结构选型

栈的实现一般可以使用数组或者链表实现，相对而言数组的结构实现更优一些。因为数组在尾上插入数据的代价比较小。

1.2 栈的实现

`stack.h`

```
typedef int STDataType;
typedef struct Stack
{
    STDataType* a;
    int top;
    int capacity;
```

```
}ST;

// 初始化栈
void STInit(ST* ps);
// 销毁栈
void STDestroy(ST* ps);

// 入栈
void STPush(ST* ps, STDataType x);
//出栈
void STPop(ST* ps);
//取栈顶元素
STDataType STTop(ST* ps);
//获取栈中有效元素个数
int STSize(ST* ps);
//栈是否为空
bool STEmpty(ST* ps);
```

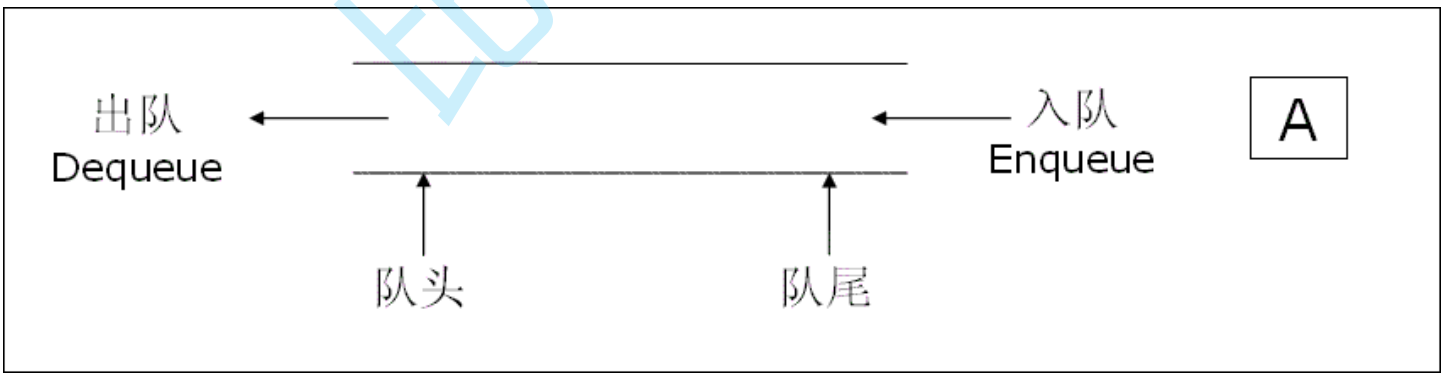
2. 队列

2.1 概念与结构

概念：只允许在一端进行插入数据操作，在另一端进行删除数据操作的特殊线性表，队列具有先进先出FIFO(First In First Out)

入队列：进行插入操作的一端称为队尾

出队列：进行删除操作的一端称为队头



> 队列底层结构选型

队列也可以数组和链表的结构实现，使用链表的结构实现更优一些，因为如果使用数组的结构，出队列在数组头上出数据，效率会比较低。

2.2 队列的实现

`Queue.h`

```c
typedef int QDataType;

//队列结点结构
typedef struct QueueNode
{
    int val;
    struct QueueNode* next;
}QNode;

//队列结构
typedef struct Queue
{
    QNode* phead;
    QNode* ptail;
    int size;
}Queue;

//初始化队列
void QueueInit(Queue* pq);
//销毁队列
void QueueDestroy(Queue* pq);

// 入队列，队尾
void QueuePush(Queue *pq, QDataType x);
// 出队列，队头
void QueuePop(Queue* pq);

//取队头数据
QDataType QueueFront(Queue* pq);
//取队尾数据
QDataType QueueBack(Queue* pq);
//队列判空
bool QueueEmpty(Queue* pq);
//队列有效元素个数
int QueueSize(Queue* pq);
```


3. 栈和队列算法题

3.1 有效的括号

https://leetcode.cn/problems/valid-parentheses/description/

3.2 用队列实现栈

https://leetcode.cn/problems/implement-stack-using-queues/description/

3.3 用栈实现队列

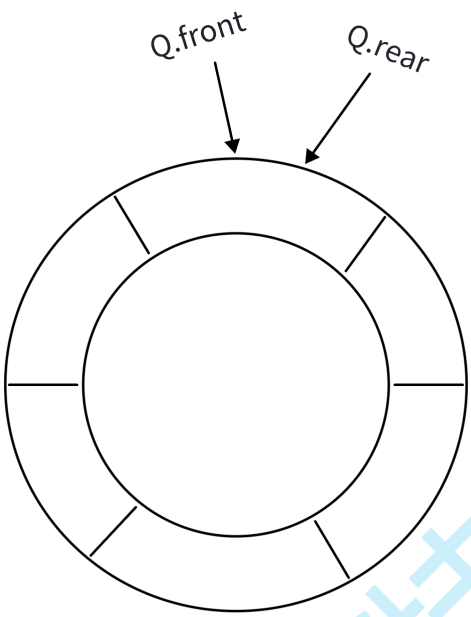
https://leetcode.cn/problems/implement-queue-using-stacks/description/

3.4 设计循环队列

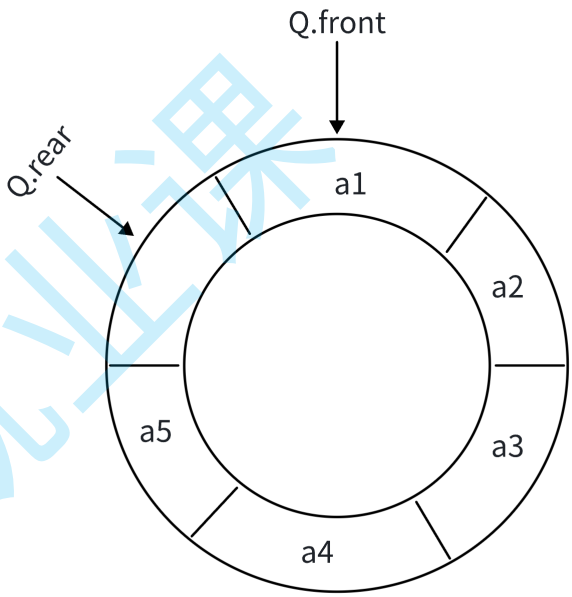
https://leetcode.cn/problems/design-circular-queue/description/

> 循环队列的概念与结构

实际中还有一种特殊的队列叫循环队列，环形队列首尾相连成环，环形队列可以使用数组实现，也可以使用循环链表实现



(a) 循环队列为空



(b) 循环队列队满

> 思考：队列满的情况下，为什么 `Q.rear` 不存储数据?

为了能使用 `Q.rear = Q.front` 来区别是队空还是队满，我们常常认为出现左图时的情况即为队满的情况

此时： `rear+1=front`