队 列

学习目标

数值队列的定义

掌握队列的操作特性

了解队列的抽象数据类型定义

掌握队列的存储结构(顺序和链式)

熟练循环队列的实现

银行排队问题

【问题】 银行个人储户的储蓄业务。

【想法】 先来先服务原则，模拟排队，储户叫号后排在队尾，窗口顺次叫号。

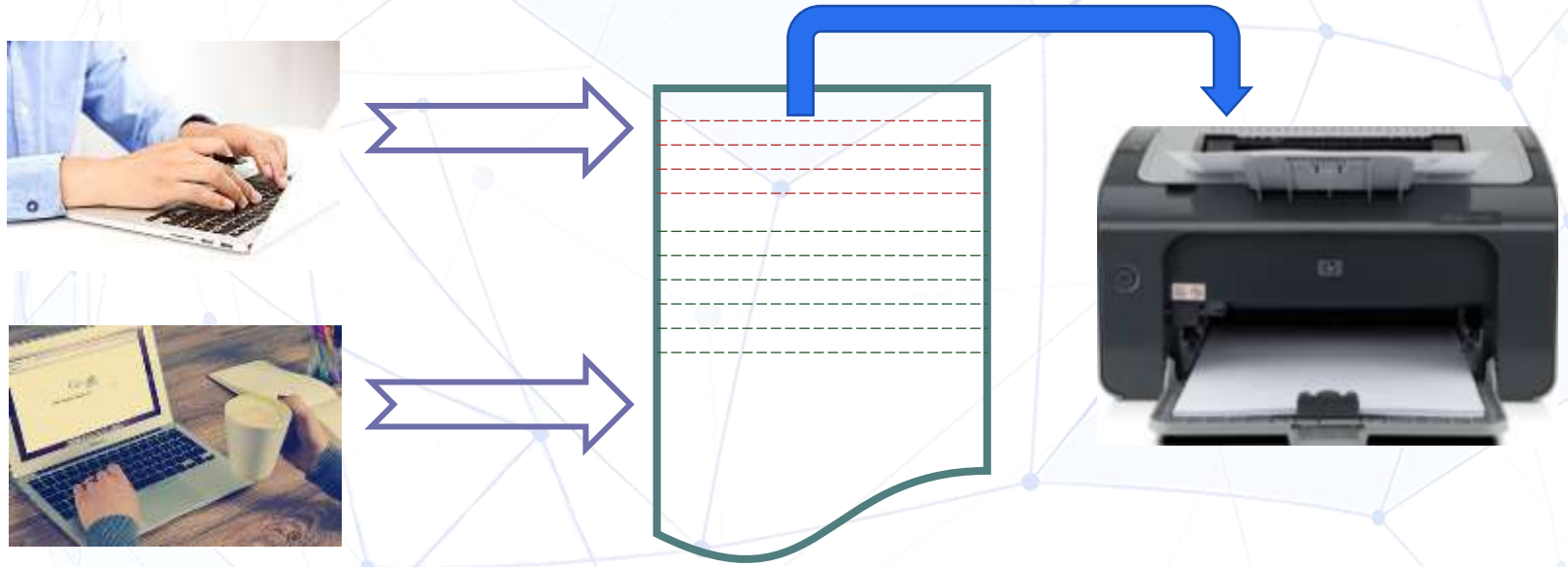
- 如何保存正在等待的储户顺序？
  - 用队列保存

在实际问题的处理过程中，有些数据具有先到先处理的特点

打印缓冲区

【问题】 多个用户共享打印机，保证打印功能。

【想法】 先来先服务原则，设置打印缓冲区，先送到缓冲区的先打印。



如何保存等待打印的文件？ ⇨ 用队列保存

在实际问题的处理过程中，有些数据具有先到先处理的特点

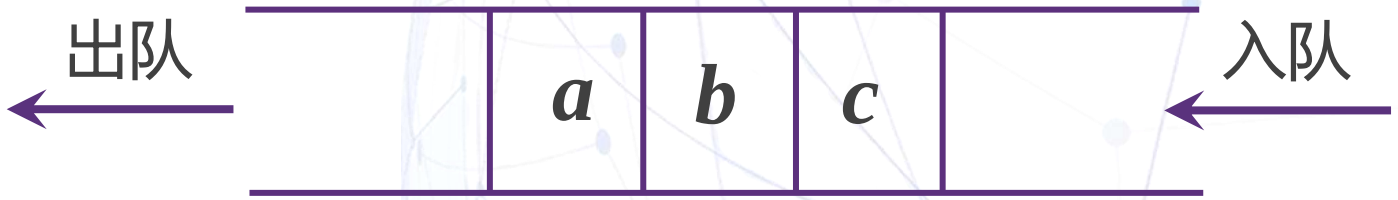
关于队列

- 什么是队列？在逻辑上有什么特点？在操作上有什么特性？
- 如何存储队列？
- 在不同的存储结构上，如何实现插入、删除、查找等基本操作？
- 在不同的存储结构上，基本操作的时空性能如何？

队列的操作特性

例：有三个元素按a、b、c的次序依次入队，且每个元素只允许进一次队，则出队序列是什么？

答：出队序列只有一种情况：$a\ b\ c$



插入：入队、进队

删除：出队

空队列：不含任何数据元素的队列

任何时候执行出队操作，一定是哪个元素呢？

队列的操作特性：先进先出（**F**irst **I**n **F**irst **O**ut，FIFO）

与栈不同，栈是一端操作，队列是两端操作

队列的抽象数据类型定义

ADT Queue

DataModel

队列中元素具有相同类型及先进先出特性，相邻元素具有前驱和后继关系

Operation

InitQueue：队列的初始化

DestroyQueue：队列的销毁

EnQueue：入队

DeQueue：出队

GetQueue：取队头元素

Empty：判空

endADT

与栈类似，队列的基本操作是确定的

队列的抽象数据类型定义

InitQueue
- 输入：无
- 功能：初始化队列，创建一个空队列
- 输出：无

DestroyQueue
- 输入：无
- 功能：销毁队列，释放队列所占用的存储空间
- 输出：无

EnQueue
- 输入：元素值x
- 功能：在队尾插入一个元素
- 输出：如果插入成功，队尾增加了一个元素；否则返回失败信息

Enqueue操作需要指明插入位置吗?

队列的抽象数据类型定义

DeQueue

输入：无

功能：删除队头元素

输出：如果删除成功，返回被删元素值；否则给出失败信息

GetQueue

输入：无

功能：读取队头元素

输出：若队列不空，返回队头元素；否则给出失败信息

Empty

输入：无

功能：判断队列是否为空

输出：如果队列为空，返回1，否则，返回0

顺序队列

- 顺序队列
- 循环队列
- 循环队列的存储结构定义
- 循环队列的实现

队列的顺序实现

基本思想：使用一个数组来存放队列中的元素，为了方便操作，还需保存队列中的队头和队尾位置

顺序实现：

- 物理模型
- 线性实现
- 循环数组实现

队列的物理模型

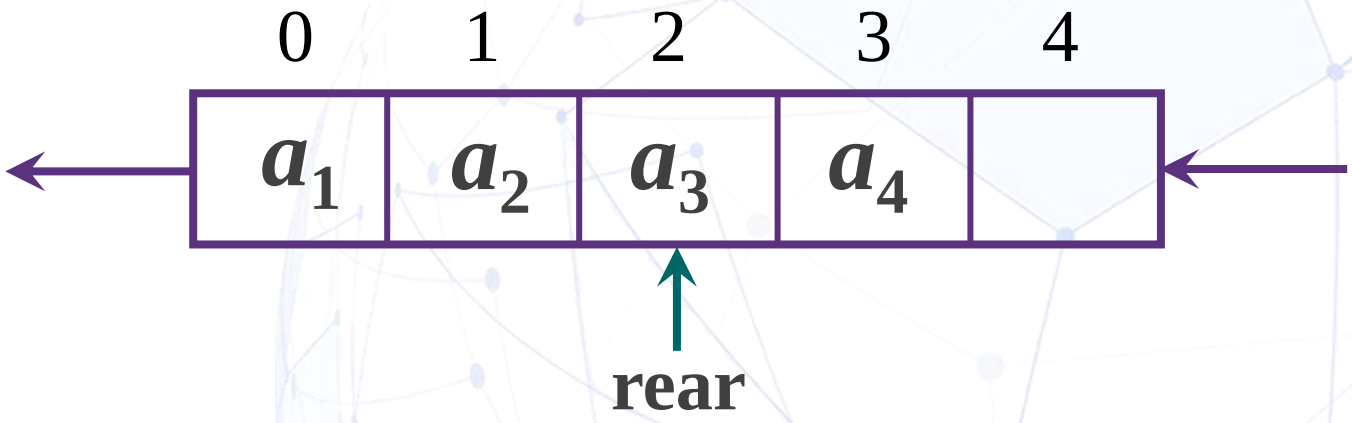
基本思想：

- 入队时元素添加到最后，队尾指示器增1
- 出队时，队列中所有元素都向前移动一个位置，逻辑上相邻，物理上也相邻

顺序队列

- 顺序队列：队列的顺序存储结构

例：$a_1a_2a_3a_4$依次入队



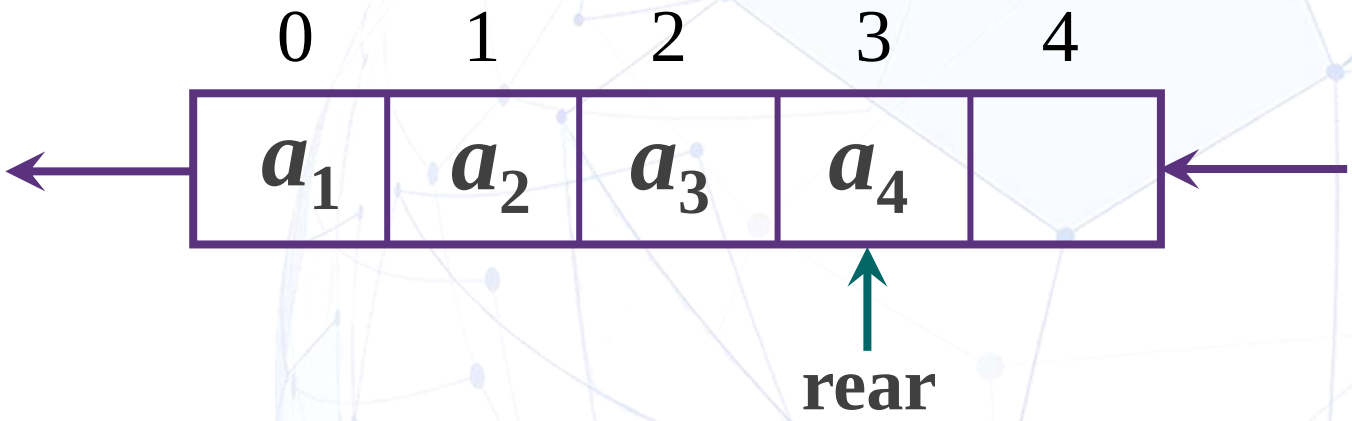
- 如何改造数组实现队列的顺序存储？
- 入队操作的时间性能？

- 如何表示队头：用数组的一端作为队头，从下标 0 处开始存放
- 如何表示队尾：设变量rear存储队尾元素所在的下标

顺序队列

- 顺序队列：队列的顺序存储结构

例：a_1a_2依次出队



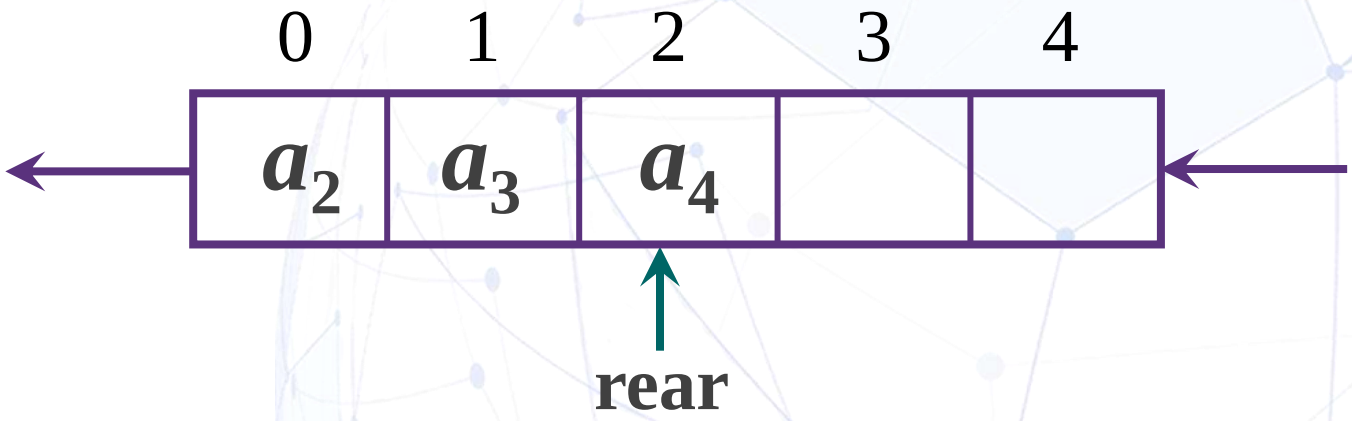
- 如何改造数组实现队列的顺序存储？

- 如何表示队头：用数组的一端作为队头，从下标 0 处开始存放
- 如何表示队尾：设变量rear存储队尾元素所在的下标

顺序队列

- 顺序队列：队列的顺序存储结构

例：a_1a_2依次出队



不实用！

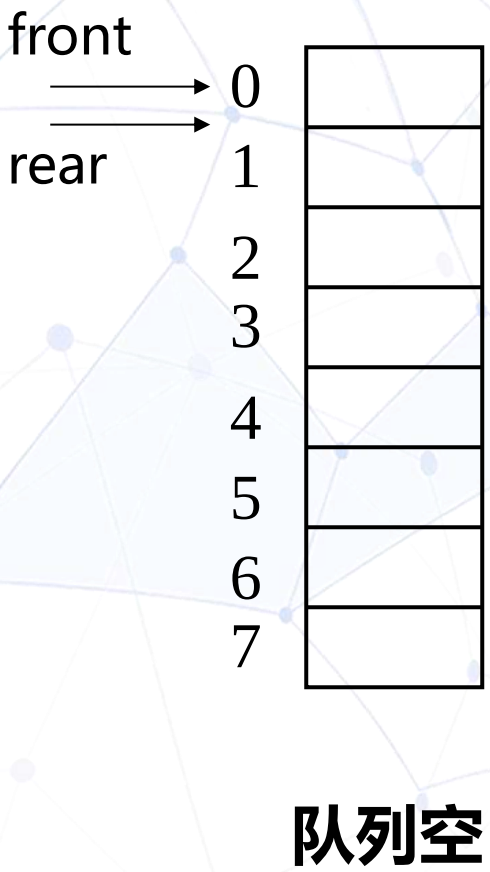
- 如何改造数组实现队列的顺序存储？
- 出队操作的时间性能？

- 如何表示队头：用数组的一端作为队头，从下标 0 处开始存放
- 如何表示队尾：设变量rear存储队尾元素所在的下标

顺序队列

如何**改进**出队操作的时间性能?

设置队头、队尾两个位置指针



队列空

线性队列

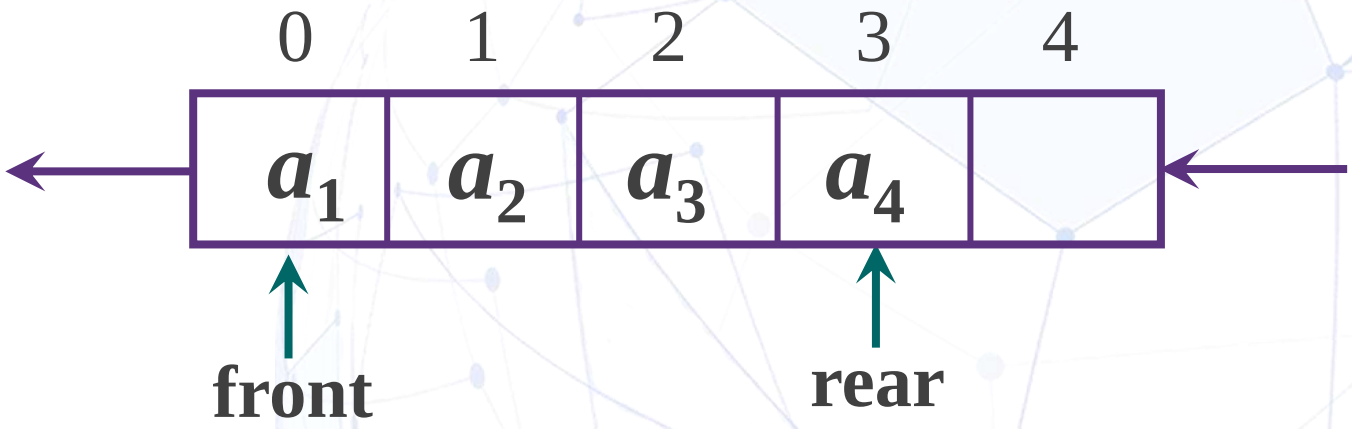
元素入队，队尾rear增加1，并将新元素放入该位置；

元素出队，从队头front获取元素，并将front加1.

顺序队列

如何**改进**出队操作的时间性能?

例：a_1a_2依次出队

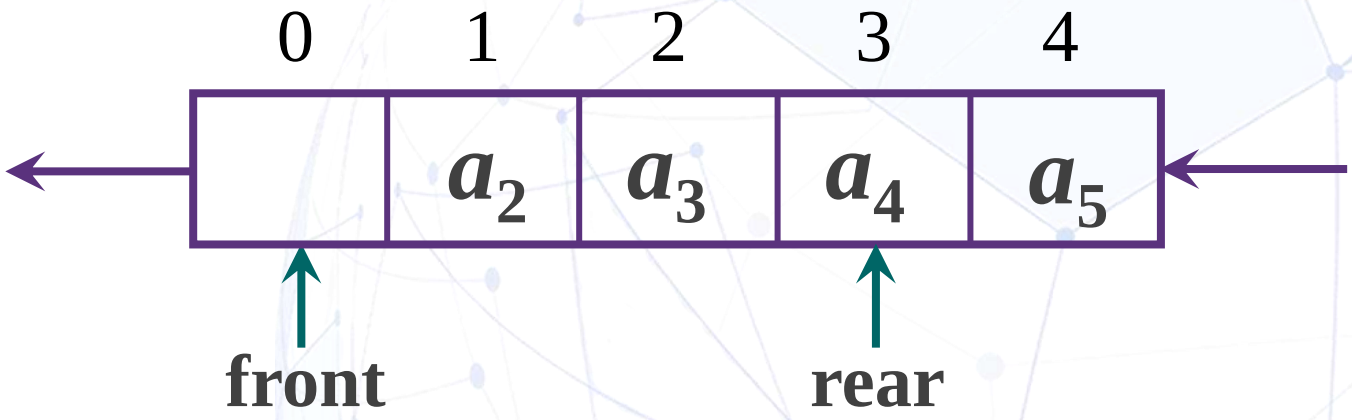


- 设置队头、队尾两个位置指针
- 入队、出队时间性能均是O(1)

顺序队列

队列的移动有什么特点?

例：a_1a_2依次出队



整个队列向数组下标较大方向移动 ⇨ 单向移动性

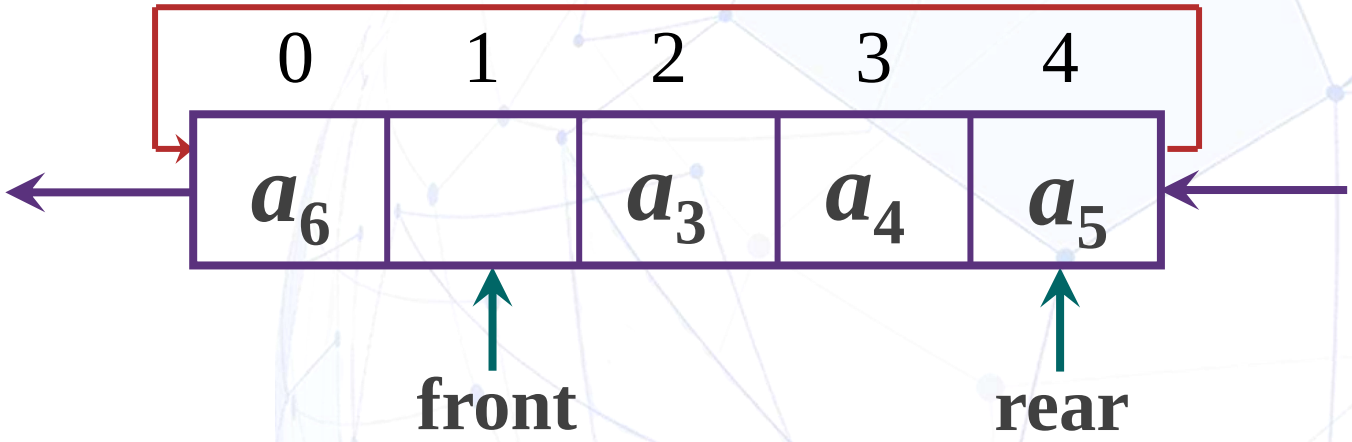
队列的单向移动性会产生什么问题?

假溢出：数组空间发生上溢，但数组的低端还有空闲空间

循环队列

如何解决假溢出呢？

如何使数组下标循环呢？



循环队列：队列采用顺序存储，并且数组是头尾相接的循环结构

```
if (rear + 1 > 4)
    rear = 0;
else
    rear++;
```

⇨ `rear = (rear + 1) % 5`

程序技巧：求模（正余数）使得数组下标循环

循环队列的类定义

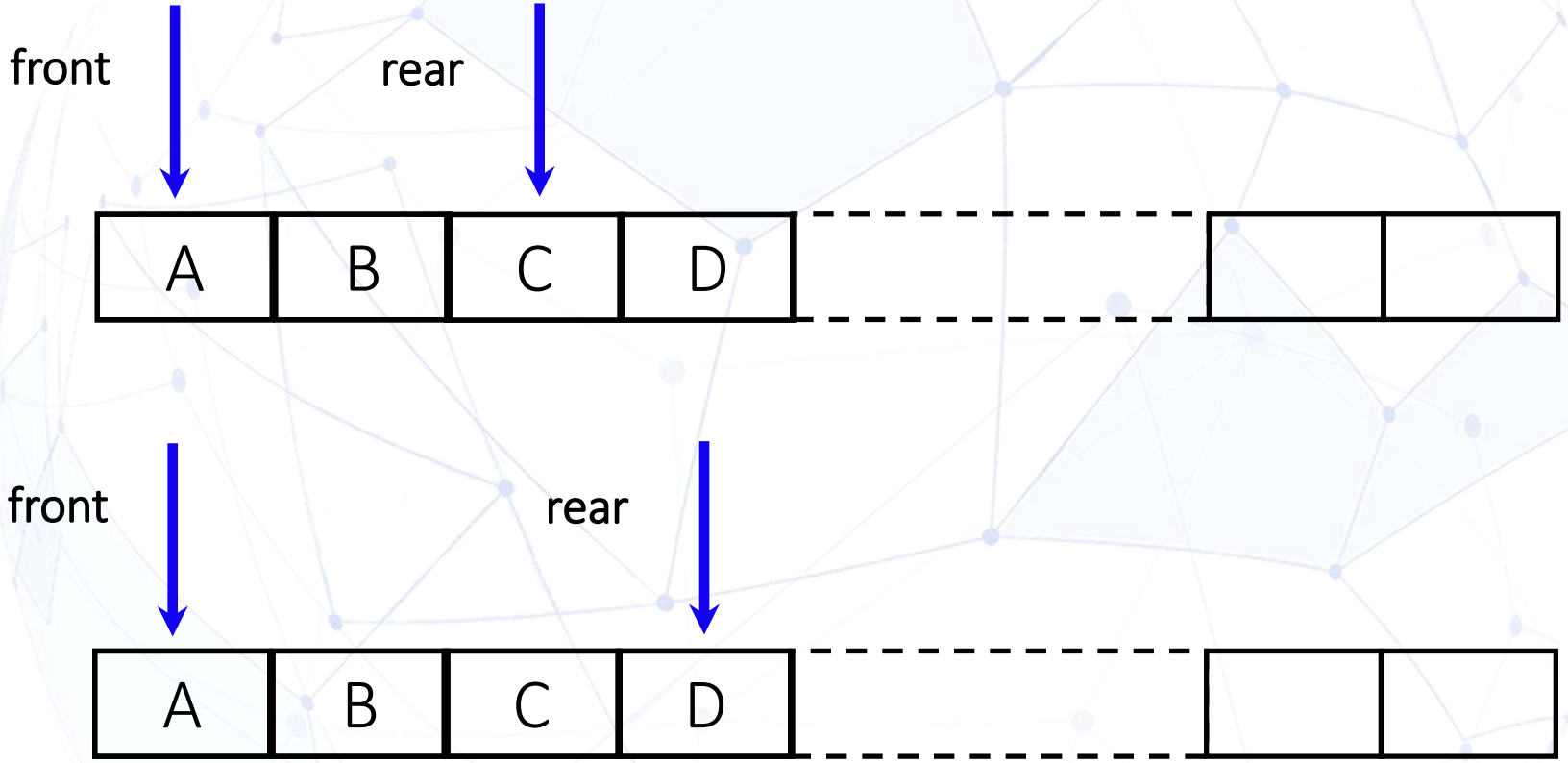
队列的抽象数据类型定义?

InitQueue：队列的初始化
DestroyQueue：队列的销毁
EnQueue：入队
DeQueue：出队
GetQueue：取队头元素
Empty：判空

```
const int QueueSize = 100;
template <typename DataType>
class CirQueue
{
public:
    CirQueue( );
    ~ CirQueue( );
    void EnQueue(DataType x);
    DataType DeQueue( );
    DataType GetQueue( );
    int Empty( );
private:
    DataType data[QueueSize];
    int front, rear;
};
```

额外说明，尾指针rear 或头指针front的特殊处理

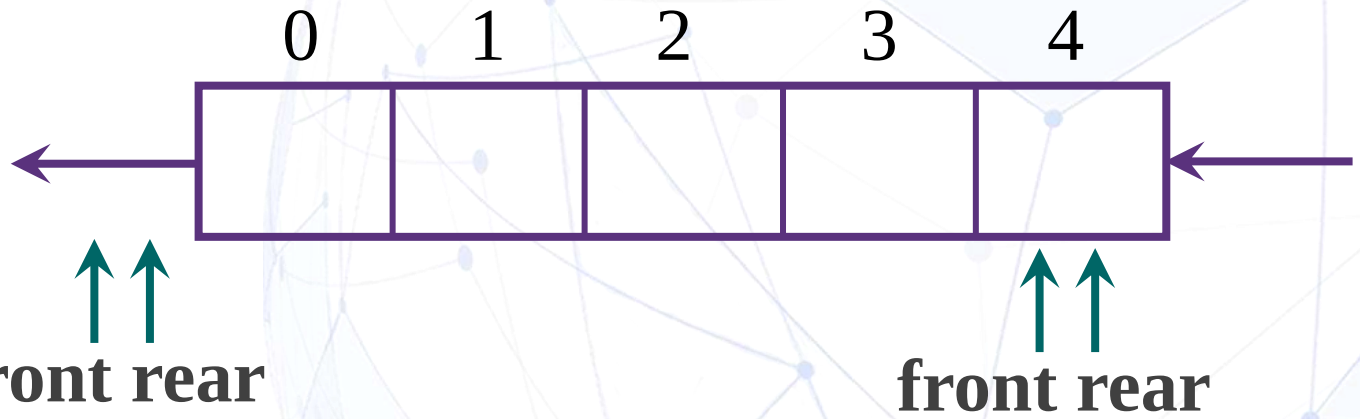
- ✓ 实指
- ✓ 虚指



同理，有些教材会约定front指向队头元素的前一个位置(即采用虚指)
本ppt采用front虚指方式

循环队列的实现——初始化

- 循环队列：队列采用顺序存储，并且数组是头尾相接的循环结构
- 设置队头、队尾两个位置指针




```
CirQueue<DataType> :: CirQueue( )
{
    front = rear = -1; //适合虚指方式
}
```

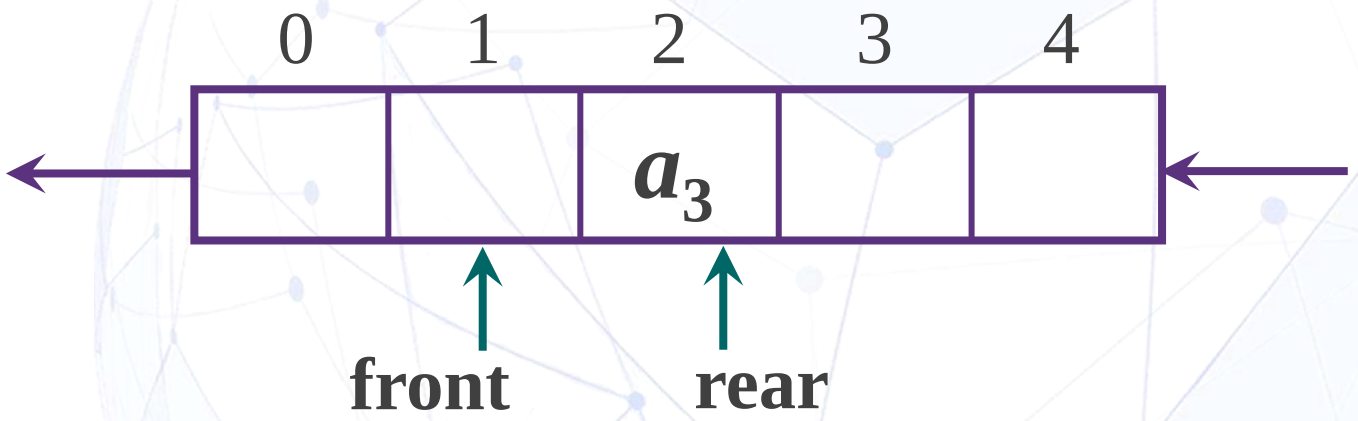
```
CirQueue<DataType> :: CirQueue( )
{
    front = rear = queuesize-1;
}
```

或者 front=0;rear=-1; 或者……

注意：初始值的不同，会影响enqueue或dequeue的代码

循环队列的实现——判空

如何判断循环队列的队空?

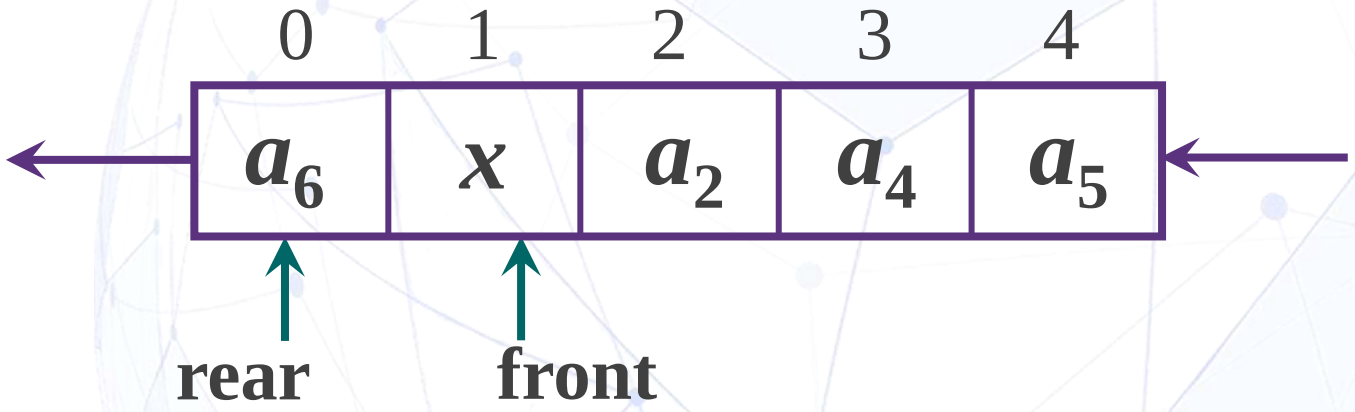


队空的判定条件： front = =rear

```
int CirQueue<DataType> :: Empty( )
{
    if (rear == front)   return 1;
    else  return 0;
}
```

队空和队满

如何判断循环队列队满?

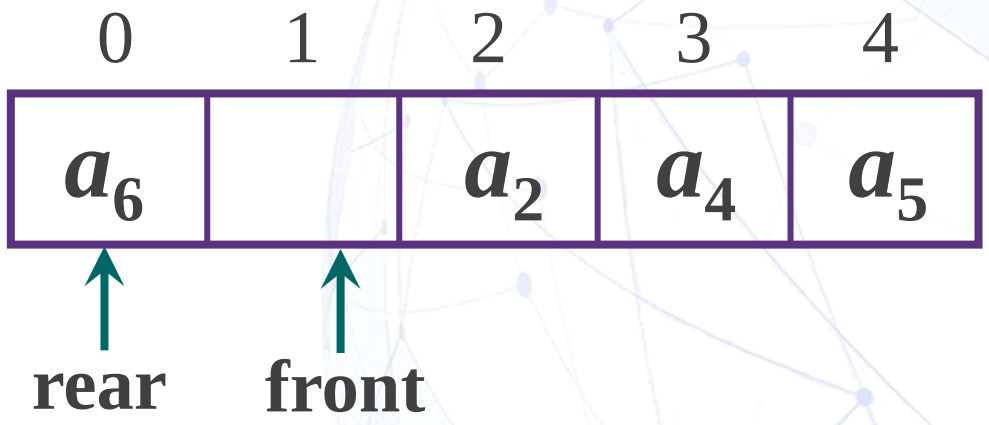


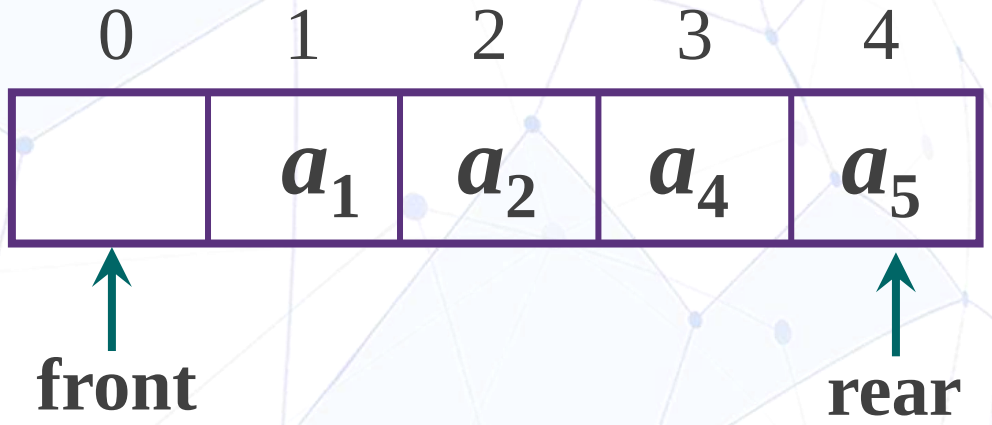
队空的判定条件：front == rear

队满的判定条件：front == rear

队空和队满

如何确定不同的队空、队满的判定条件?



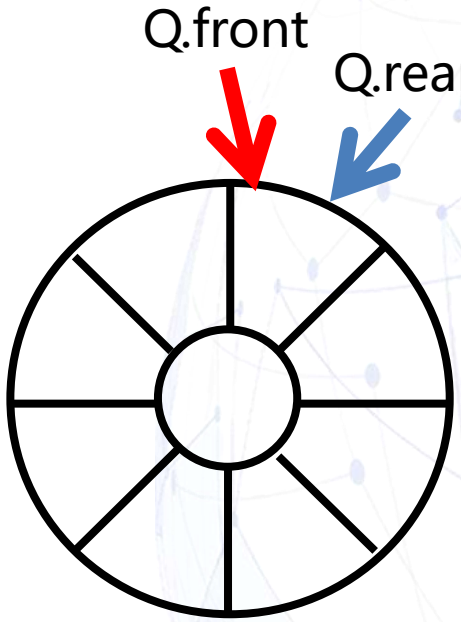


队空的判定条件：front == rear

队满的判定条件：front == rear

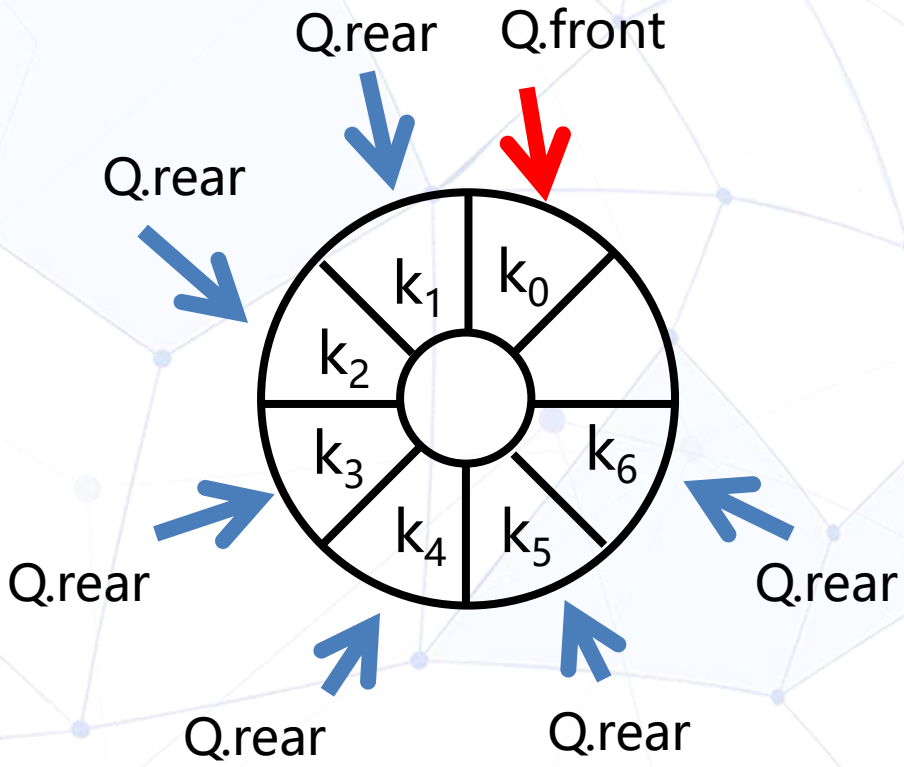
(rear+1) % QueueSize == front

⇨ 数组中有一个空闲单元



队列空：

Q.rear == Q.front



队列满：

(Q.rear+1) mod QueueSize == Q.front

其它解决判断队列空或满的方法

引入一个变量的方法

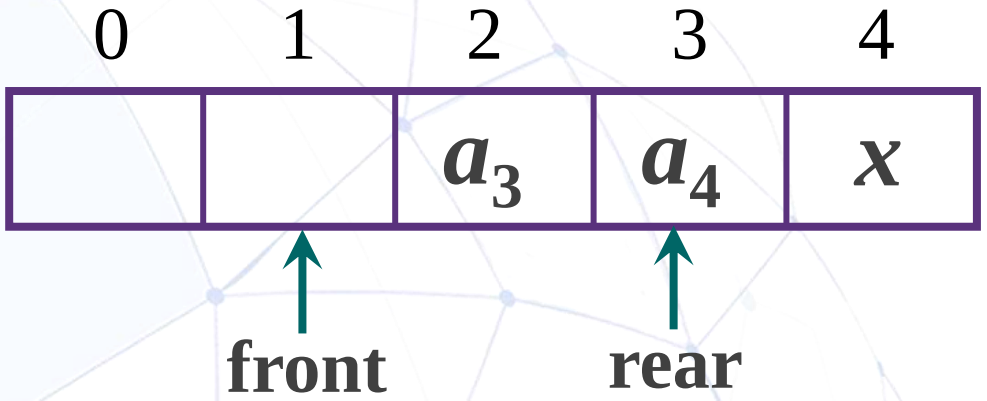
一个Boolean型变量表示rear是否刚刚到达front的前面；

一个int型变量记录队列中当前元素的个数

一个变量指示最后一次完成的动作是入队还是出队

……

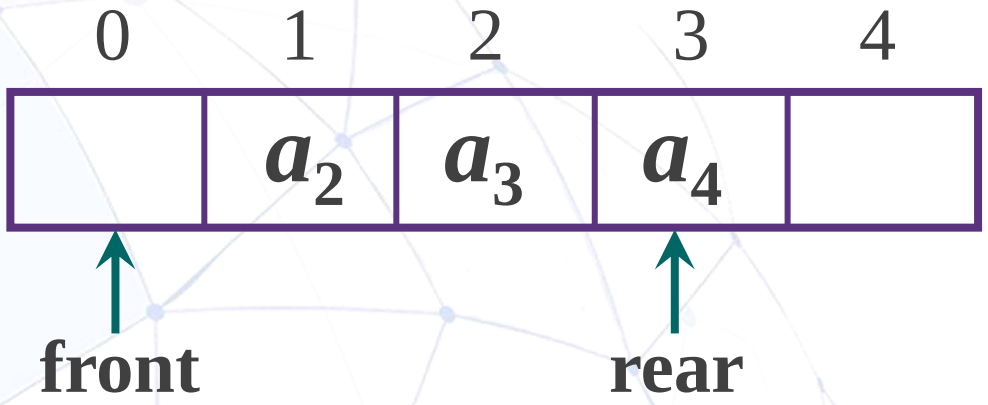
循环队列的实现——入队




```
template <typename DataType>
void CirQueue<DataType> :: EnQueue(DataType x)
{
    if ((rear + 1) % QueueSize == front) throw "上溢";
    rear = (rear + 1) % QueueSize;                    //队尾指针在循环意义下加1
    data[rear] = x;                                   //在队尾处插入元素
}
```

时间复杂度是多少?

循环队列的实现——出队




```
template <typename DataType>
DataType CirQueue<DataType> :: DeQueue( )
{
    if (rear == front) throw "下溢";
    front = (front + 1) % QueueSize;          //队头指针在循环意义下加1
    return data[front];                       //读取并返回出队前的队头元素
}
```

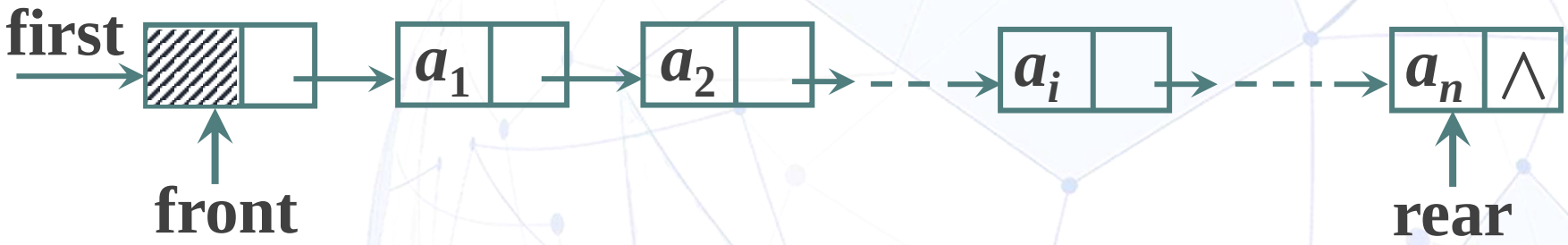
取队头元素的实现?

链队列

- 链队列的存储结构定义
- 链队列的实现——入队
- 链队列的实现——出队
- 链队列的实现——取队头元素
- 链队列的实现——销毁

链队列的存储结构定义

链队列：队列的链接存储结构



P：用链表的哪一端作为队头？哪一端作为队尾？

A：链头作为队头，出队时间为$O(1)$

链尾作为队尾，入队时间为$O(n)$ ⇨ 设置队尾指针rear

P：链队列需要加头结点吗？

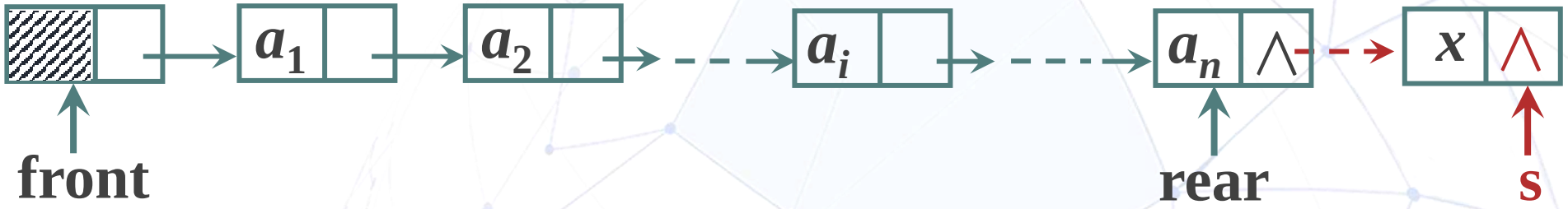
链队列的类定义

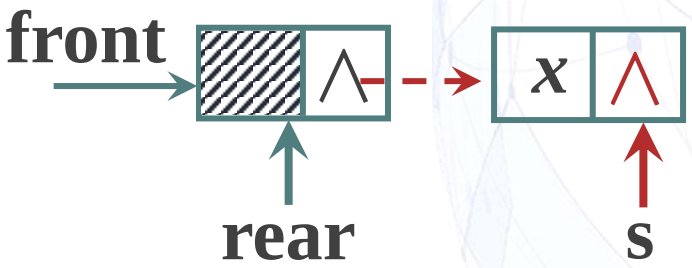
队列的抽象数据类型定义?

InitQueue：队列的初始化
DestroyQueue：队列的销毁
EnQueue：入队
DeQueue：出队
GetQueue：取队头元素
Empty：判空

```
template <typename DataType>
class LinkQueue
{
public:
    LinkQueue( );
    ~LinkQueue( );
    void EnQueue(DataType x);
    DataType DeQueue( );
    DataType GetQueue( );
    int Empty( );
private:
    Node<DataType> *front, *rear;
};
```

链队列的实现——入队





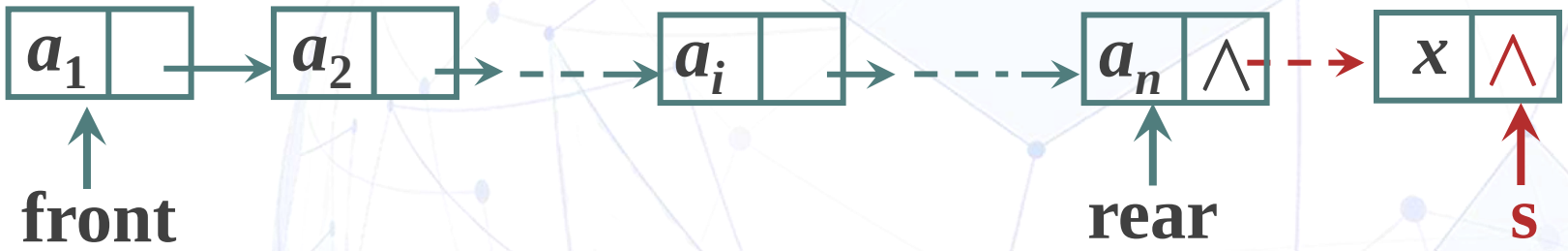

```
void LinkQueue<DataType> :: EnQueue(DataType x)
{
    Node<DataType> *s = nullptr;
    s = new Node<DataType>;                    //申请结点s
    s->data = x; s->next = nullptr;
    rear->next = s; rear = s;
}
```

带头节点的队列处理比较方便

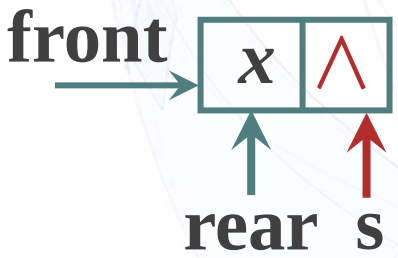
时间复杂度是多少?

链队列的实现——入队

没有头结点会怎样?



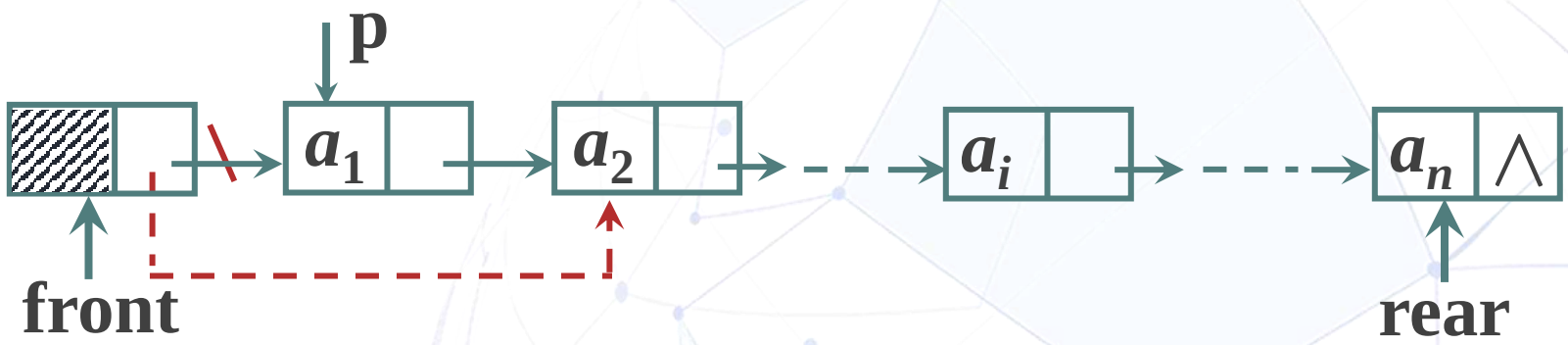
rear->next = s; rear = s;

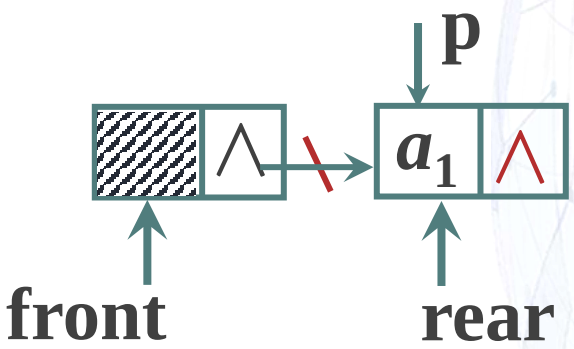


无头结点需要分情况写代码

rear = s; front = s;

链队列的实现——出队



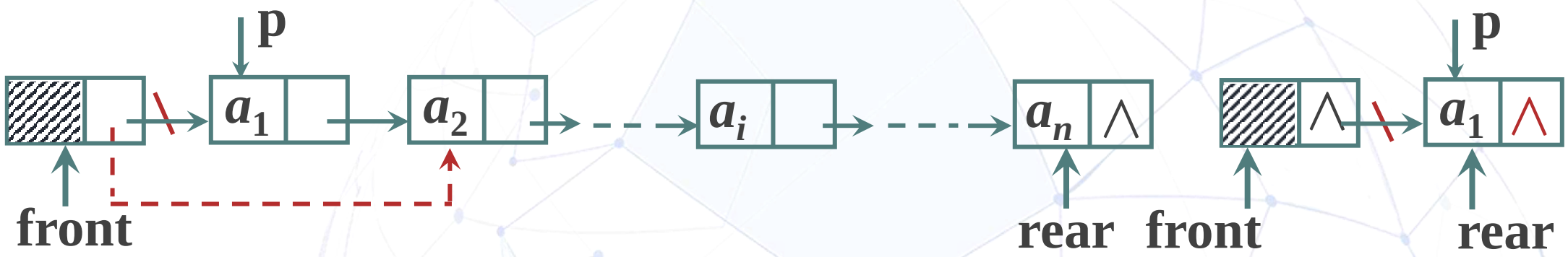


如何判断边界情况?

```
if (p->next == nullptr)
    rear = front;
```

考虑边界情况：队列中只有一个元素?

链队列的实现——出队




```
DataType LinkQueue<DataType> :: DeQueue( )
{
    DataType x;
    Node<DataType> *p = nullptr;
    if (rear == front) throw "下溢";
    p = front->next; x = p->data;
    front->next = p->next;
    if (p->next == nullptr) rear = front;
    delete p;
    return x;
}
```

取队头元素的实现?

变种的栈和队列结构

- 双端队列

  限制插入和删除在线性表的两端进行两个底部相连的栈

- 超队列

  一种删除受限的双端队列：删除只允许在一端进行，而插入可在两端进行

- 超栈

  一种插入受限的双端队列，插入只限制在一端而删除允许在两端进行

课堂小结

队列的定义和逻辑结构

队列的实现(顺序和链式)

队列的假溢出

循环队列的队空和队满