数据结构与算法

疫情面前，让我们一起努力！

1

单选题 2分

为解决计算机主机与打印机间速度不匹配问题，通常设一个打印数据缓冲区。主机将要输出的数据依次写入该缓冲区，而打印机则依次从该缓冲区中取出数据。该缓冲区的逻辑结构应该是（　）。

A 队列

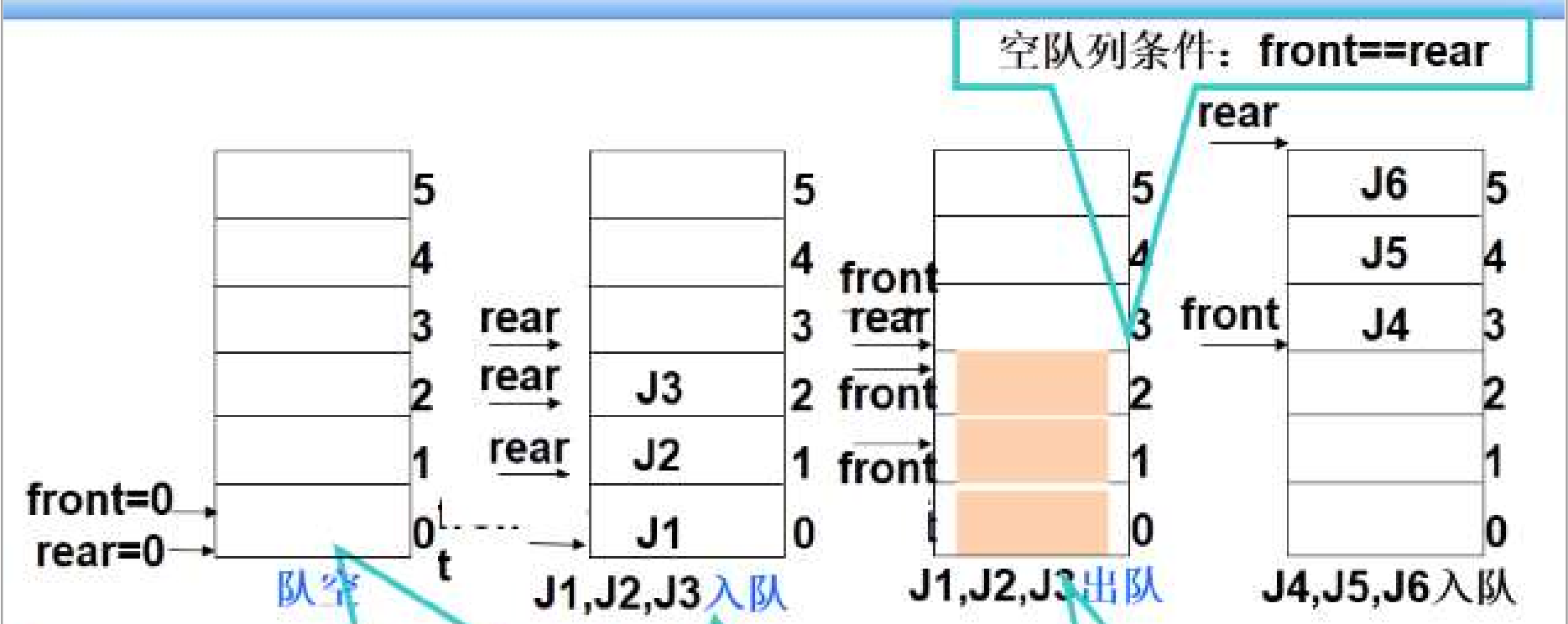
B 栈

C 线性表

D 有序表

循环队列的顺序存储结构和实现

（用一维数组实现sq[M]）

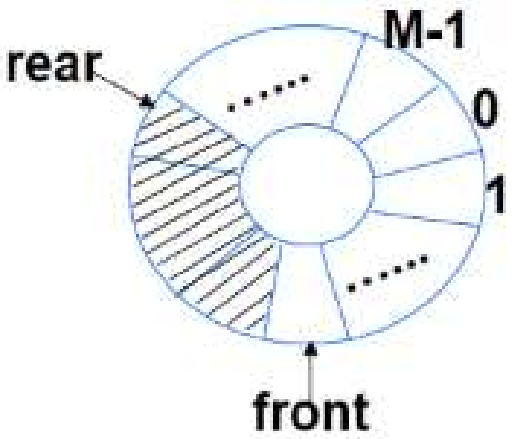


设两个指针front,rear, 约定：
front指示队头元素；
rear指示队尾元素的下一个位置；
初值front=rear=0

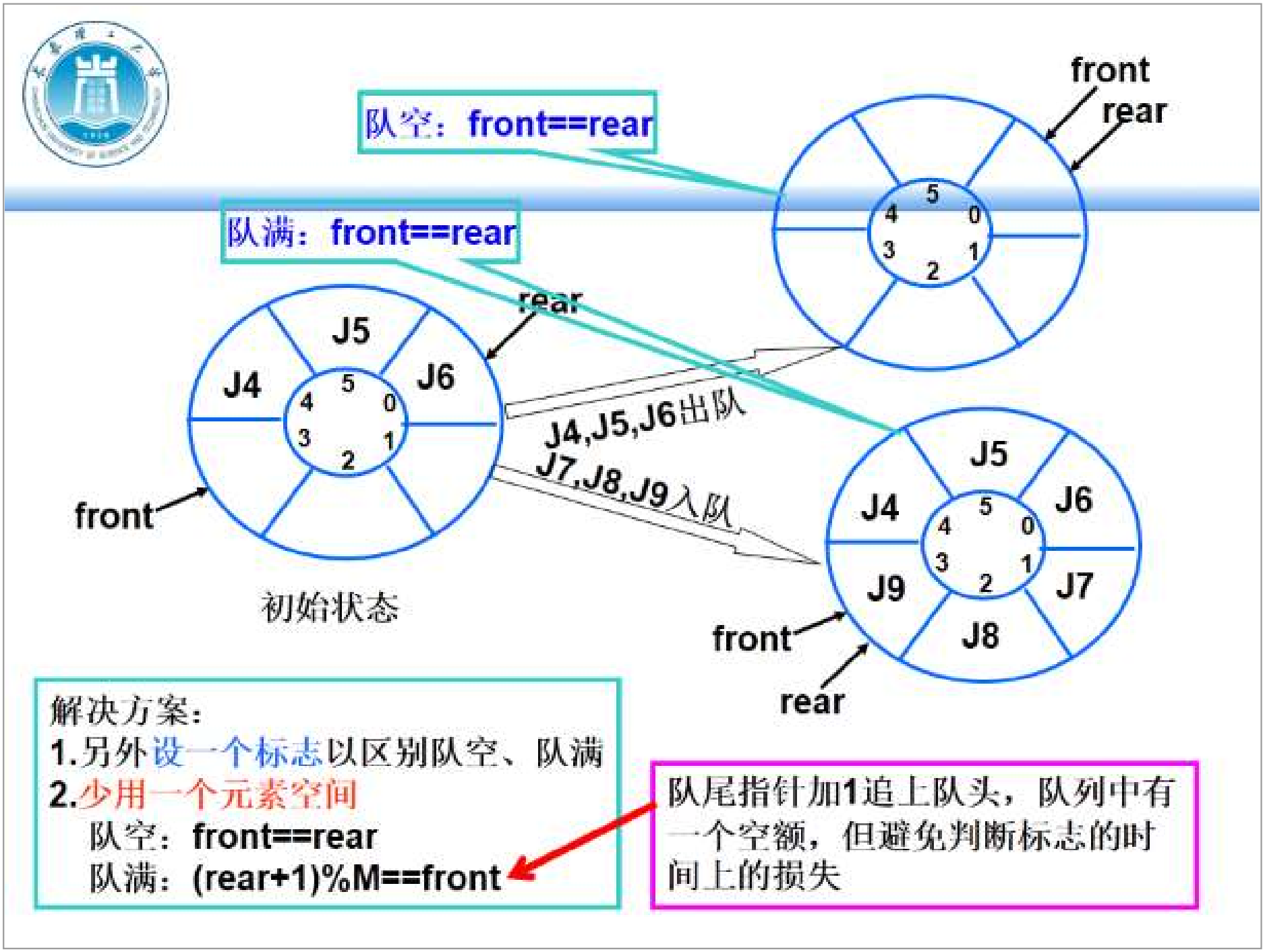
入队列：
sq[rear++]=x;

出队列：x=sq[front++];

- 存在问题
  - 设数组维数为M，则：
    - 当front=0,rear=M时，再有元素入队发生溢出——真溢出
    - 当front≠0,rear=M时，再有元素入队发生溢出——假溢出
- 解决方案
  - 队首固定，每次出队后剩余元素向下移动——浪费时间
  - 循环队列
    - 基本思想：把队列设想成环形，让sq[0]接在sq[M-1]之后，若rear+1==M, 则令rear=0;



- 实现：利用“模”运算
- 入队：
```
rear=(rear+1)%M;
sq[rear]=x;
```
- 出队：
```
front=(front+1)%M;
x=sq[front];
```


队空：front==rear
front
rear
5
4
0
3
1
2
队满：front==rear
rear
J5
J4
J6
front
J4,J5,J6出队
J7,J8,J9入队
初始状态
J9
J7
J8
front
rear
解决方案：
1.另外设一个标志以区别队空、队满
2.少用一个元素空间
队空：front==rear
队满：(rear+1)%M==front
队尾指针加1追上队头，队列中有一个空额，但避免判断标志的时间上的损失

循环队列的顺序存储结构和实现

循环队列类型说明：

```
#define QueueSize 100
 typedef Struct{
      int front;
      int rear;
     QElemType   *base; 或QElemType
data[QueueSize];
     }SqQueue;
```

循环队列的基本操作的算法描述

1. 构造空队列

```
void InitQueue(SqQueue &Q){
   （1）
Q.base=(QElemType*)malloc(QueueSize*sizeof(QEl
emType));
   （2）if(!Q.base) exit(OVERFLOW);
   （3）Q.front=Q.rear=0;
 }
```

循环队列的基本操作的算法描述

2. 返回队列长度

```
int QueueLength(SqQueue Q){
  （1） Return (Q.rear-Q.front+ QueueSize )%
QueueSize;
}
```

循环队列的基本操作的算法描述

3. 入队

```
Status EnQueue(SqQueue &Q,QElemType e){
  （1）if((Q.rear+1)% QueueSize ==Q.front )
  （2）return ERROR;
  （3） Q.base[Q.rear]=e;
  （4）Q.rear=(Q.rear+1)% QueueSize;
   （5）  return OK;
}
```

循环队列的基本操作的算法描述

4. 出队

```
Status DeQueue(SqQueue &Q,QElemType
&e) {
  if(Q.front==Q.rear) return ERROR;
  e=Q.base[Q.front];
  Q.front=(Q.front+1)% QueueSize;
  return OK;
}
```

链队列

队列的链式存储结构简称为链队列，它是限制仅在表头删除和表尾插入的单链表。显然仅有单链表的头指针不便于在表尾做插入操作，为此再增加一个尾指针，指向链表的最后一个结点。于是，将链队列的类型LinkQueue定义为一个结构类型：

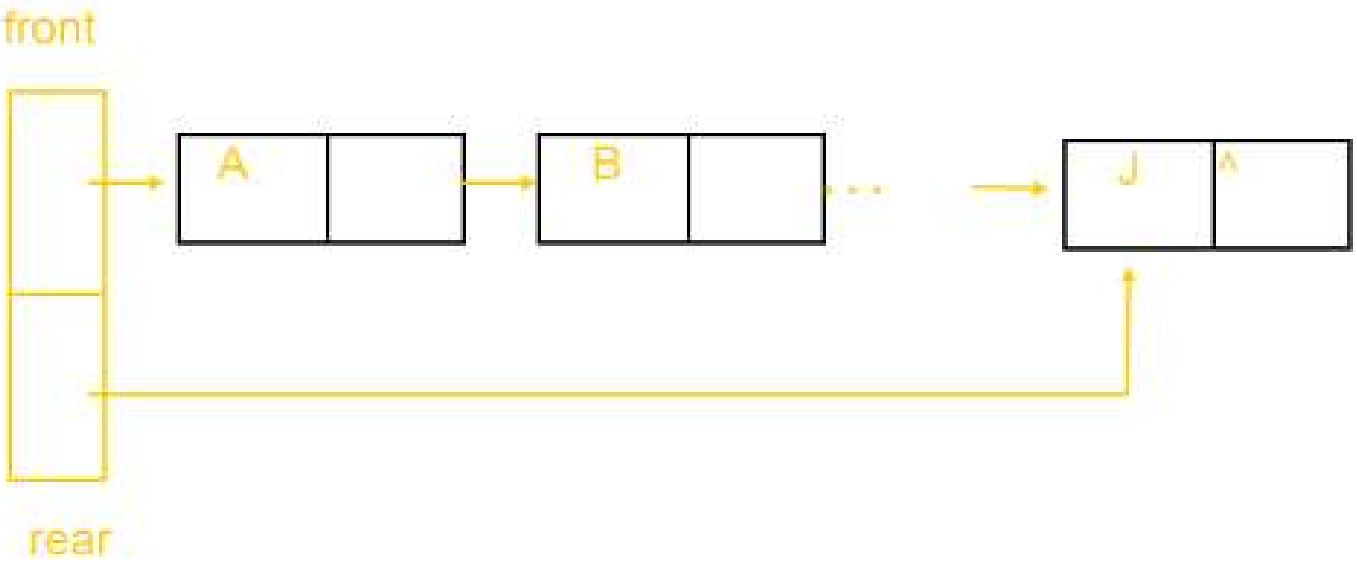
```
typedef struct QNode{
    QElemType  data;
    struct QNode *next;
}QNode,*QueuePtr;
```

```
typedef struct{
    QueuePtr  front;
    QueuePtr  rear;
}LinkQueue;
```

链队列

头指针和尾指针封装在一个结构中的链队列

如图所示为头指针和尾指针封装在一个结构中的链队列。



链队列上实现的基本运算

1.构造一个空队列

```
void InitQueue(LinkQueue  &Q)
{
  （1）
Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));
  （2）if(!Q.front) exit(OVERFLOW);
  （3） Q.front->next=NULL;
}
```

链队列上实现的基本运算

2.销毁队列

```
void DestoryQueue(LinkQueue &Q)
{
   while(Q.front)
    {
          （1）Q.rear=Q.front->next;
          （2）free(Q.front);
          （3）Q.front=Q.rear;
    }
}
```

链队列上实现的基本运算

3.入队

```
void EnQueue(LinkQueue  &Q, QElemType  e){
    （1）p=(QueuePtr )malloc(sizeof(QNode));
    （2）p–>data=e;
    （3）p–>next=null;
    （4）Q.rear->next=p;  Q.rear=p;
}
```

链队列上实现的基本运算

4. 出队

```
Status DeQueue(LinkQueue &Q, QElemType &e) {
    （1）if(Q.front==Q.rear)  return ERROR;
    （2）p=Q.front->next;
    （3）e=p->data;
    （4）Q.front->next=p->next;
    （5）if(Q.rear==p)  Q.rear=Q.fi
    （6）free(p);
}//end DeQueue
```

注意：在出队算法中，一般只需修改队头指针。但当原队中只有一个结点时，该结点既是队头也是队尾，故删去此结点时亦需修改尾指针，且删去此结点后队列变空。

队列的总结

上述过程中，我们可以总结：

队列中增加元素时，rear++；

队列中减少元素时，front++。

换句话说，队列的基础操作的关键是rear、front指针。

队列特性：先进先出

如何比较循环队列和链队列？

17

多选题 3分

关于循环队列和链队列的比较，哪种说法是正确的？

A 从时间上说，其基本操作都是常数阶的。

B 循环队列事先申请好空间，使用期间不释放。

C 链队列每次申请和释放结点不需要时间开销。

D 循环队列需要一个固定的长度，容易产生空间浪费。

今天的课程即将开始，本节课的学习重点：

队列的应用

20

中秋舞会

21

假设在周末舞会上，男士们和女士们进入舞厅时，各自排成一队。跳舞开始时，依次从男队和女队的队头各出一人配成舞伴。若两队初始人数不相同，则较长的那一队中为配对者等待下一轮舞曲。现要求写一算法模拟上述舞伴配对问题。

22

- 设置两个队列分别存放男士和女士入队者
- 假设男士和女士的记录存放在一个数组中作为输入，然后依次扫描该数组的各元素，并根据性别来决定是进入男队还是女队。
- 当这两个队列构造完成之后，依次将两队当前的队头元素出队来配成舞伴，直至某队列变空为止。
- 此时，若某队仍有等待配对者，则输出此队列中排在队头的等待者的姓名，此人将是下一轮舞曲开始时第一个可获得舞伴的人。

23