栈和队列

3.1 栈和队列的基本概念

3.1.1 栈的基本概念

栈的定义：
一种只能在一端进行插入或删除操作的线性表，允许进行插入或删除操作的一端成为栈顶。
另一端成为栈底，固定不变。

1. 特点：**先进后出（FILO）**
2. 存储结构：顺序栈和链式栈

3.1.2 队列的基本概念

1. 定义：一种操作受限的线性表，仅允许在表的一端进行插入，在另一端进行删除，可进行插入的一端为**队尾**，可进行删除的一端为**对头**。
2. 特点：**先进先出（FIFO）**
3. 存储结构：顺序队，链队

3.1.3 连续输入输出

栈：输入序列与输出序列顺序相反。
队列：输入序列与输出序列顺序相同。

3.1.4 非连续输入输出

1. 判断是合法出栈序列的方法：出栈序列中每个元素后面所有比它小的元素要组成一个递减序列。
2. 计算合法出栈序列的数量：$\frac{1}{n+1}C_{2n}^{n}$

3.2 存储结构、算法、应用

3.2.1 结构体定义

1.顺序栈定义

```
typedef struct {
    int data[maxSize];
```

```
    int top;
} SqStack;
```

2.链节点定义

```
typedef struct Node {
    int data;
    struct LNode *next;
} LNode;
```

链栈采用链表来存储栈，用带头结点的单链表来座位存储体。

3.顺序队列定义

```
typedef struct {
    int data[maxSize];
    int front;
    int rear;
} SqQueue;
```

4.链队定义

（1）队结点类型定义

```
typedef struct QNode {
    int data;
    struct QNode *next;
} QNode;
```

（2）链队类型定义

```
typedef struct {
    QNode *front;
    QNode *rear;
}LiQueue;
```

3.2.2 顺序栈

1.顺序栈的要素：

两个特殊状态、两个操作

状态	栈空状态	栈满状态	非法状态

描述	st.top==-1	st.top=maxSize-1	栈满时继续入栈造成上溢； 栈空时继续出栈造成下溢。

操作	**进栈**	**出栈**
代码表示	++(st.top); st.data(st.top)=x;	x=st.data[st.top]; --(st.top);

2.初始化栈的代码：

```
void initStack(SqStack &st) {
    st.top = -1;
}
```

3.判断栈空:

```
int isEmpty(SqStack &st) {
    if(st.top == -1)
        return -1;
    else
        return 0;
}
```

4.进栈代码:

```
int push(SqStack &st, int x) {
    if(st.top == maxSize - 1)
        return 0;
    ++(st.top);
    st.data[st.top] = x;            //可以简写为 st.data[++st.top] = x;
    return 1;
}
```

5.出栈代码:

```
int pop(SqStack &st, int &x) {
    if(st.top == -1)
        return -1;
    x = st.data[st.top];
    --(st.top);                     //可以简写为 x = st.data[st.top--];
    return 1;
}
```

3.2.3 链栈

1.链栈的要素：

两个特殊状态、两个操作

状态	栈空状态	栈满状态
描述	lst->next == NULL	不存在

操作	进栈	出栈

代码表示	p->next = lst->next;
	lst->next = p;
		p = lst->next; x = p->data;
		lst->next = p->next;
		free(p);

2.初始化代码：

```
void initStack(LNode *&lst){
    lst = (LNode*)malloc(sizeof(LNode));        //申请内存，制造头节点
    lst->next = NULL;
}
```

3.判断栈空代码：

```
int isEmpty(LNode *lst) {
    if(lst->next == NULL)
        return 1;
    else
        return 0;
}
```

4.进栈代码：

```
void push(LNode *lst, int x) {
    LNode *p;
    p = (LNode*)malloc(sizeof(LNode));
    p->next = NULL;
    p->data = x;
    p->next = lst->next;
    lst->next = p;
}
```

5.出栈代码：

```
int pop(LNode *lst, int &x) {
    LNode *p;
    if(lst->next == NULL)
        return 0;
    p = lst->next;
    x = p->data;
    lst->next = p->next;
    free(p);
    return 0;
}
```

3.2.4 顺序队

1.循环队列

解决了假溢出。将数组弄成一个环，让rear和front沿着环走。

2.循环队列的要素：

状态	队空	队满
描述	qu.rear == qu.front;	(qu+1)%maxSize == qu.front;

3.两个操作：

操作	进队	出队
代码表示	qu.rear=(qu.rear+1)%maxSize; qu.data[qu.rear]=x;	qu.front=(qu.front+1)%maxSize; x=qu.data[qu.front];

4.初始化队列算法：

```
void initQueue(SqQueue &qu) {
    qu.front = qu.rear = 0;
}
```

5.判断队空算法：

不论对首、队尾指针指向数组中哪个位置，只要二者重合，即为空队。

```
int isQueueEmpty(SqQueue qu) {
    if(qu.front == qu.rear)
        return 1;
    else
        return 0;
}
```

6.进队：

```
int enQueue(SqQueue &qu, int x) {
    if((qu.rear+1)%maxSize == qu.front)
        return 0;
    qu.rear = (qu.rear + 1) % maxSize;
    qu.data[qu.rear] = x;
    return 1;
}
```

7.出队：

```
int deQueue(SqQueue &qu, int x) {
    if(qu.front == qu.rear)
        reuturn 0;
    qu.front = (qu.front + 1) % maxSize;
    x = qu.data[qu.front];
    return 1;
}
```

3.2.5链队

1.链队的要素：

状态	队空	队满
描述	lqu->rear == NULL; 或 lqu->front == NULL;	不存在

操作	进队	出队

代码表示	lqu->rear->next = p;
	lqu->rear = p;
		p = lqu->front;
		lqu->front=p->next;
		x=p->data;
		free(p);

2.初始化链队：

```
void initQueue(LiQueue *&lqu) {
    lqu = (LiQueue*)malloc(sizeof(LiQueue));
    lqu->front = lqu->rear = NULL;
}
```

3.判断队空：

```
int isQueueEmpty(LiQueue *lqu) {
    if(lqu->rear == NULL || lqu->front == NULL)
        return 1;
    else
        return 0;
}
```

4.入队算法：

```
void enQueue(LiQueue *lqu, int x) {
    QNode *p = (QNode*)malloc(sizeof(QNode));
    p->data = x;
    p->next = NULL;
    if(lqu->rear == NULL)
        lqu->front = lqu->rear = p;
    else {
        lqu->rear->next = p;
        lqu->rear = p;
    }
```

```
}
```

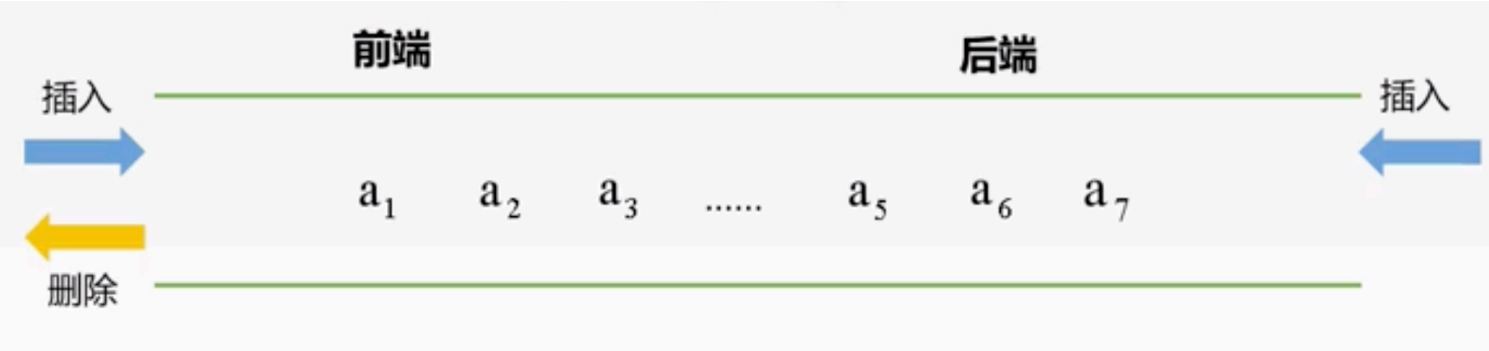
5.出队算法：

```
void deQueue(LiQueue *lqu, int x) {
    QNode *p;
    if(lqu->rear == NULL)
        return 0;
    else
        p = lqu->front;
    if(lqu->front == lqu->rear)
        lqu->front = lqu->rear = NULL;
    else
        lqu->front = lqu->front->next;
    x = p->data;
    free(p);
    return 1;
}
```

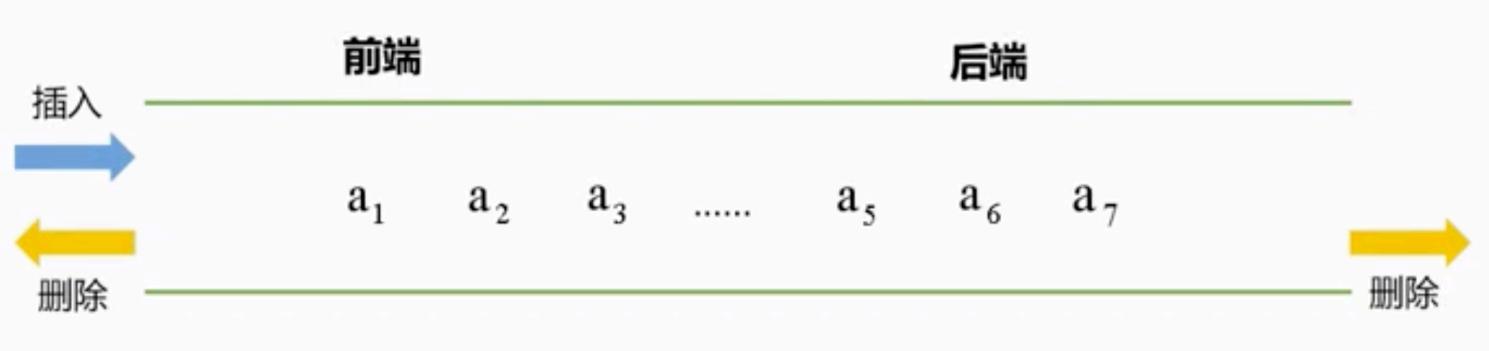
3.2.6 双端队列

1.双端序列：允许两端进行插入和删除的队列。

2.输出受限的双端队列：两端都可进行插入，只有一端可以进行删除。



3.输入受限的双端队列：两端都可进行删除，只有一端可以进行插入。



3.2.7 栈的应用

1.括号匹配算法：

（1）初始一个栈，顺序读入括号。

（2）若是右括号，则与栈顶元素进行匹配

- 若匹配，则弹出栈顶元素并进行下一个元素
- 若不陪陪，则该序列不合法

（3）若是左括号，则入栈

（4）若全部元素遍历完毕，占中非空则序列不合法。

2.表达式求值

（1）算术表达式三种形式：前缀式、中缀式、后缀式。

（2）算法：

a.若为'('，入栈；

b.若为')'，则一次把栈中的预算福加入后缀表达式，直到出现'('，再从栈中删除'('；

c.若为‘+’，‘–’，‘*’，‘/’：

- 栈空，入栈；
- 栈顶元素为'('，入栈；
- 高于栈顶元素优先级，入栈；
- 否则，一次弹出栈顶运算符，直到一个优先级比它低的运算符或'('为止；

d.遍历完成，若栈非空则依次弹出所有元素

3.3 特殊矩阵的压缩存储

1.对称矩阵：

$k = i(i-1)/2+j-1$ $(i \geqslant j)$

$k = j(j-1)/2+i-1$ $(i<j)$

2.三角矩阵：

$k = i(i-1)/2+j-1$ $(i \geqslant j)$

$k = n(n-1)/2$ $(i<j)$

3.三对角矩阵：

$k = 2i+j-3$

4.稀疏矩阵：

三元组。例：

i	j	value
0	0	1

2	3	8
3	2	2

- 稀疏矩阵压缩存储后，失去了随机存储的特征。