Chapter 3 栈和队列

书P131. 3.2

证明：用**反证法**。假设存在 i<j<k，使得$p_j<p_k<p_i$成立。则说明输入序列中的顺序是：…p_j … p_k … p_i …。

若要p_i先出栈，则p_j,p_k必须先于p_i压入栈中，按照后进先出的特征，出栈的顺序就只能是…p_i…p_k…p_j…，这与输出序列是…p_i …p_j … p_k…相矛盾。因此假设不成立。

2009考研统考题：

设栈S和队列Q的初始状态均为空，元素a, b, c, d, e, f, g依次进入栈S。若每个元素出栈后立即进入队列Q，且7个元素出队的顺序是b, d, c, f, e, a, g，则栈S的容量至少是（C）

A. 1　　B. 2　　C. 3　　D. 4

2013考研统考题：

一个栈的入栈序列为1, 2, 3, …, n，其出栈序列是p_1, p_2, p_3, …, p_n。若p_2=3，则p_3可能取值的个数是（C）

A. n-3　　B. n-2　　C. n-1　　D. 无法确定

书P132. 3.14: 已知A[n]为整数数组，写出以下递归算法：

（1）求数组A中的最大整数

```
int maxValue(int A[ ], int n) {
    if (n==1) return A[0];
    else {
        int temp=maxValue(A, n-1);
        return (temp>A[n-1]) ? temp : A[n-1];
    }
}
```

（2）求n个整数的和

```
int Sum(int A[ ], int n) {
    if (n==1) return A[0];
    else return Sum(A, n-1)+A[n-1];
}
```

书P133. 3.22: 只使用队尾指针和计数器的循环队列

循环队列类定义：

```
template <class T>
class CQueue<T> {
private:
    int rear, length;        //队尾指针、队列长度
    T *elements;             //队列元素数组
    int maxSize;             //最大元素个数
public:
    CQueue ( int sz= 10 );
    ~CQueue ( ) { delete [ ] elements; }
    bool EnQueue (T x);              //进队(入队)
    bool DeQueue (T& x );            //出队
    bool GetFront (T& x);            //取队头元素
    void MakeEmpty ( ) { length = 0; }
    bool IsEmpty ( ) const  { return length == 0; }
    bool IsFull ( ) const  { return length == maxSize; }
};
```

```
template <class T>
bool CQueue<T>::EnQueue ( T x) {    //入队算法
    if  (IsFull ( ))  return false;
    elements[rear] = x;
    rear = (rear+1) % maxSize;
    length++;
    return true;
}

 template <class T>
 bool CQueue<T> :: DeQueue(T& x) {    //出队算法
     if ( IsEmpty ( )) return false;
     x=elements[(rear-length+maxSize)%maxSize];
     length--;
     return true;
 }
```

书P134. 3.23： 实现带标志的循环队列。

带标志的循环队列类定义：

```
template <class T>
class CTQueue<T> {
private:
    int rear, front, tag;   //队尾指针、队头指针、标志
    T *elements;            //队列元素数组
    int maxSize;            //最大元素个数
public:
    CTQueue ( int sz= 10 );
    ~CTQueue ( ) { delete [ ] elements; }
    bool EnQueue (T x);             //进队(入队)
    bool DeQueue (T& x );           //出队
    bool GetFront (T& x);           //取队头元素
    void MakeEmpty ( ) { front=rear=tag= 0; }
    bool IsEmpty ( ) const  { return front==rear && tag==0; }
    bool IsFull ( ) const  { return front==rear && tag==1; }
};
```

```
template <class T>
bool CTQueue<T>::EnQueue ( T x) {    //入队算法
    if  (IsFull ( ))  return false;
    elements[rear] = x;
    rear = (rear+1) % maxSize;
    tag=1;          //表示队列不空
    return true;
}

template <class T>
bool CTQueue<T> :: DeQueue(T& x) {    //出队算法
    if ( IsEmpty ( )) return false;
    x=elements[front];
    front = (front+1) % maxSize;
    tag=0;          //表示队列不满
    return true;
}
```

Chapter 4 数组

书P183. 4.2 对称矩阵的压缩存储（$1 \le i, j \le n$）

（2）存上三角部分：

若 $i \le j$，$k = n + (n-1) + (n-2) + \cdots + (n-i+2) + j-i$

$= (2n-i+2)*(i-1) / 2 + j-i$

$= (2n-i)*(i-1) / 2 + j-1$

若$i > j$，$k = (2n-j)*(j-1) / 2 + i-1$

（3）存下三角部分：

若 $i \ge j$，$k = 1 + 2 + \cdots + i-1 + j-1$

$= (i-1)*i / 2 + j-1$

若 $i < j$，$k = (j-1)*j / 2 + i-1$

书P183. 4.3

将A、B两个下三角矩阵存到n行n+1列矩阵C

$$A[i][j]=\begin{cases}C[i][j], i \geq j\\ 0, i<j\end{cases}$$

$$B[i][j]=\begin{cases}C[j][i+1], i \geq j\\ 0, i<j\end{cases}$$

或写成：

$$C[i][j]=\begin{cases}A[i][j], i \geq j\\ B[j-1][i], i<j\end{cases}$$

书P183. 4.4 三对角矩阵按列优先的压缩存储

（$1 \le i, j \le n$ ， $j-1 \le i \le j+1$ ）

（1）$k = 2 + 3(j-2) + i-j+1$

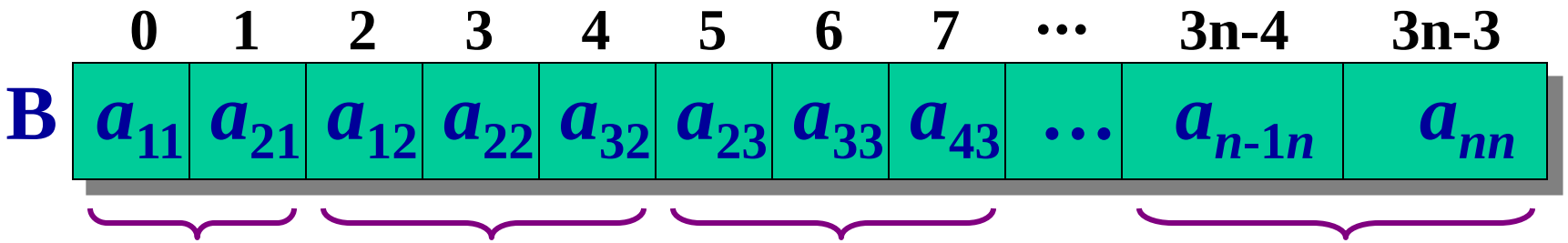
$= 2j + i - 3$

（2） $2j + j-1-3 \le k \le 2j + j+1-3$

$3j - 4 \le k \le 3j - 2$

故，$j = \lfloor (k + 1) / 3 \rfloor + 1$ 或 $j = \lceil (k+2) / 3 \rceil$

$i = k - 2j + 3$



补充：（软考真题）

设W为一个二维数组，其中每个数据元素W[i][j]占用6个字节，行下标i从0到8，列下标j从2到5，则二维数组W的数据元素共占用_A_个字节；W中第6行的元素和第4列的元素共占用_B_个字节；若按行优先顺序存放二维数组W，其起始地址的字节号为100，则二维数组W的最后一个数据元素的起始地址的字节号为_C_，数据元素W[3][4]的起始地址号为_D_，而数据元素W[2][2]的起始地址号与当按列优先顺序存放时数据元素_E_的起始地址相同。

A （1）480	（2）192	（3）216 √	（4）144
B （1）78	（2）72 √	（3）66	（4）84
C （1）310 √	（2）311	（3）184	（4）185
D （1）179	（2）178	（3）184 √	（4）185
E （1）W[0][5]	（2）W[2][8]	（3）W[5][2]	（4）W[8][2] √

解：行下标从0-8，共9行；

列下标从2-5，共4列。

总存储空间需9×4×6=216个字节

$$W=\begin{pmatrix} w_{02} & w_{03} & w_{04} & w_{05} \\ w_{12} & w_{13} & w_{14} & w_{15} \\ w_{22} & w_{23} & w_{24} & w_{25} \\ w_{32} & w_{33} & w_{34} & w_{35} \\ w_{42} & w_{43} & w_{44} & w_{45} \\ w_{52} & w_{53} & w_{54} & w_{55} \\ w_{62} & w_{63} & w_{64} & w_{65} \\ w_{72} & w_{73} & w_{74} & w_{75} \\ w_{82} & w_{83} & w_{84} & w_{85} \end{pmatrix}$$

补充：

已知$A_{n\times n}$是稀疏矩阵，试从空间和时间角度比较采用二维数组和三元组顺序表两种不同存储结构实现 $\sum_{i=1}^{n} a_{ii}$ 运算的优缺点。

解：

	空间	时间
二维数组：	$O(n^2)$	$O(n)$
三元组顺序表：	$O(t)$	$O(t)$

- 当非零元素的个数 *t* 和 *n* 同数量级时，两者时间一样，三元组顺序表的空间优于二维数组。
- 当非零元素的个数 *t* 和 *n*n* 同数量级时，两者的空间都为$O(n^2)$，但采用二维数组的时间更优。