4.3

$$A=\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad B=\begin{bmatrix} b_{11} & & & \\ b_{21} & b_{22} & & \\ \vdots & \vdots & \ddots & \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$C=\begin{bmatrix} a_{11} & b_{11} & b_{21} & \cdots & b_{n-1\,1} & b_{n1} \\ a_{21} & a_{22} & b_{12} & \cdots & b_{n-1\,2} & b_{n2} \\ a_{31} & a_{32} & a_{33} & & b_{n-1\,3} & b_{n3} \\ \cdots & \cdots & \cdots & & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_{nn} \end{bmatrix}$$

$$A\,[i]\,[j]=\begin{cases} C\,[i]\,[j], & i \geqslant j \\ 0, & i<j \end{cases} \qquad B\,[i]\,[j]=\begin{cases} C\,[j]\,[i+1], & i \geqslant j \\ 0, & i<j \end{cases}$$

4.8

i	0	1	2	3
s	a	a	a	b
next(i)	-1	0	1	2

i	0	1	2	3	4	5	6
t	a	b	c	a	b	a	a
next(i)	-1	0	0	0	1	2	1

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
r	a	b	c		a	a	b	b	a	b	c	a	b	a	a	c	b	a	c	b	a
next(i)	-1	0	0	0	0	1	1	2	0	1	2	3	1	2	1	1	0	0	1	0	0

4.12

```
void Saddle(int** A, int m, int n) {
    for (int i = 0; i < m; i++) {
        int min = 0;
        for (int j = 1; j < n; j++) {
            if (A[i][j] < A[i][min]) min = j;
        }
        bool found = true;
        for (int k = 0; k < m; k++) {
            if (A[k][min] > A[i][min]) found = false;
        }
        if (found) {
            cout << "Saddle point: " << "A[" << i << "][" << min << "] = "
                 << A[i][min] << endl;
        }
```

```
    }
}
```

该函数的时间复杂度

$$O(m \times \max\{m, n\})$$

4.15

```
void frequency(string& s, char ch[], int freq[], int& size){
    if (s.length() == 0) {
        cout << "The string is empty." << endl;
        return;
    }
    int i = 0;
    size = 0;
    while (i < s.length()) {
        int j = 0;
        while (j < size) {
            if (s[i] == ch[j]) {
                freq[j]++;
                break;
            }
            j++;
        }
        if (j == size) {
            ch[j] = s[i];
            freq[j] = 1;
            size++;
        }
        i++;
    }
}

int main()
{
    string s;
    cout << "Enter a string: ";
    getline(cin, s);
    char ch[100];
    int freq[100];
    int size = 0;
    frequency(s, ch, freq, size);
    for (int i = 0; i < size; i++) {
        cout << ch[i] << ": " << freq[i] << endl;
    }
    return 0;
}
```

4.16

(1)

```
template<class T> class GenListNode;
template<class T> class GenList;

template<class T>
struct Items {
    bool mark;
    int utype;
    union {
        char* Lname;
        T value;
        GenListNode<T>* hlink;
    } info;
    Items() :mark(false), utype(0), info.Lname('\0') {  }
    Items(Items<T>& RL) { mark = RL.mark; utype = RL.utype; info = RL.info; }
};

template<class T>
class GenListNode {
    friend class GenList;
public:
    GenListNode() :mark(false), utype(0), info.Lname('\0'), tlink(NULL) {   }
    GenListNode(GenListNode<T>& RL) { mark = RL.mark; utype = RL.utype; info =
RL.info; tlink = RL.tlink; }
private:
    bool mark;
    int utype;
    union {
        char* Lname;
        T value;
        GenListNode<T>* hlink;
    } info;
    GenListNode<T>* tlink;
};

template<class T>
class GenList {
public:
    GenList();
    ~GenList() { Remove(first); }
    bool Head(Items& x);
    bool Tail(GenList<T>& lt);
    GenListNode<T>* First() { return first; }
    GenListNode<T>* Next(GenListNode<T>* elem) { return elem->tlink; }
    void Copy(const GenList<T>& R) { first = Copy(R.first); }
    int Depth() { return Depth(first); }
    int Length() { return Length(first->tlink); }
    friend istream& operator>>(istream& in, GenList<T>& L);
    void Traverse();
private:
    GenListNode<T>* first;
    GenListNode<T>* Copy(GenListNode<T>* ls);
```

```
    int Depth(GenListNode<T>* ls);
    int Length(GenListNode<T>* ls);
    bool Equal(GenListNode<T>* s, GenListNode<T>* t);
    void Remove(GenListNode<T>* ls);
    void CreateList(istream& in, GenListNode<T>* ls, SeqList<T>& L1,
SeqList<GenListNode<T>*>& L2);
    void Traverse(GenListNode<T>* ls);
};
```

(2)

```
template<class T>
void GenList<T>::Traverse() {
    Traverse(first);
}

template<class T>
void GenList<T>::Traverse(GenListNode<T>* ls) {
    if (ls != NULL) {
        ls->mark = true;
        if (ls->utype == 0) cout << ls->info.Lname << "(";
        else if (ls->utype == 1) {
            cout << ls->info.value;
            if (ls->tlink != NULL) cout << ",";
        }
        else {
            if (!ls->info.hlink->mark) {
                Traverse(ls->info.hlink);
            }
            else {
                cout << ls->info.hlink->info.Lname;
                if (ls->tlink != NULL) cout << ",";
            }
        }
        Traverse(ls->tlink);
    }
    else cout << ")";
}
```

(3)

```
template<class T>
void GenList<T>::Traverse() {
    Stack<GenListNode<T>*> st;
    GenListNode<T>* ls = first;
    while (ls != NULL) {
        ls->mark = true;
        if (ls->utype == 2) {
            if (!ls->info.hlink->mark) {
                st.Push (ls->tlink);
                ls = ls->info.hlink;
            }
            else {
                cout << ls->info.hlink->info.Lname;
```

```
                if (ls->tlink != NULL) {
                    cout << ",";
                    ls = ls->tlink;
                }
            }
        }
        else {
            if (ls->utype == 0) cout << ls->info.Lname << "(";
            else {
                cout << ls->info.value;
                if (ls->tlink != NULL) cout << ",";
            }
            if (ls->tlink == NULL) {
                cout << ")";
                if (!st.Empty()) {
                    st.Pop(ls);
                    if(ls != NULL) cout << ",";
                    else cout << ")";
                }
                else ls = ls->tlink;
            }
            else ls = ls->tlink;
        }
    }
}
```