数据结构习题课

2.38

设有一个双向循环链表，每个结点中除有prc,data和next三个域外，还增设了一个访问频度域freq。在链表被起作用之前，频度域freq的值均初始化为零，而每当对链表进行一次LOCATE（L，x）的操作后，被访问的节点（即元素值等于x的结点）中的频度域freq的值便增1，同时调整链表中结点之间的次序，使其按访问频度非递增的次序顺序排列，以便始终保持被频繁访问的节点总是靠近表头节点。试编写符合上述要求的LOCATE操作的算法。

```
DuLinkList ListLocate_DuL(DuLinkList &L,ElemType e)
{
DuLinkList p,q;
p=L->next;
while(p!=L && p->data!=e) p=p->next;
if(p==L) return NULL;
else{
p->freq++;
// 删除结点
p->pre->next=p->next;
p->next->pre=p->pre;
// 插入到合适的位置
q=L->next;
while(q!=L && q->freq>p->freq) q=q->next;
if(q==L){
p->next=q->next;
q->next=p;
p->pre=q->pre;
q->pre=p;
}
else{
// 在 q 之前插入
p->next=q->pre->next;
q->pre->next=p;
p->pre=q->pre;
q->pre=p;
}
return p;
}
}
 2.39 至 2.40 题中，稀疏多项式采用的顺序存储结构 SqPoly 定义为
typedef struct {
int coef;
int exp;
} PolyTerm;
typedef struct { //多项式的顺序存储结构
PolyTerm *data;
int last;
} SqPoly;
```

跳马

国际象棋之跳马程序

问题描述:

假设国际象棋棋盘有5*5共25个格子。设计一个程序,使棋子从初始位置（棋盘格编号为1的位置)开始跳马,能够把棋盘的格子全部走一遍,每个格子只允许走一次。要求:

1) 输出一个解(用二维数组来记录马跳的过程,即[步号,棋盘格编号],左上角为第一步起点)，2)求总共有多少解

棋盘格编号为:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

```
#include <iostream>
using namespace std;

const int m = 5, n = 5;
int countN = 0;
int trace[m][n] = { 0 };
int changedir[][2] = { {1,2},{2,1},{1,-2},{2,-1},{-1,2},{-2,1},{-1,-2},{-2,-1}
};

void printT()
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << trace[i][j] << '\t';
        }
        cout << endl;
    }
    cout << "------------" << endl;
}

void visit(int x, int y, int step)
{
    trace[x][y] = step;
    //printT();
    if (m*n == step)
    {
        countN++;
        printT();
        return;
    }
    int nextx, nexty;
    for (int i = 0; i < 8; i++)
    {
        nextx = x + changedir[i][0];
        nexty = y + changedir[i][1];
        if (nextx < 0 || nextx>(m-1) || nexty < 0 || nexty>(n-1)|| trace[nextx]
[nexty]!=0)
        {
            continue;
        }
        visit(nextx, nexty, step + 1);
        trace[nextx][nexty] = 0;
    }
}
void main()
{

    int firstX , firstY ;
    cout << "输入起始位置（棋盘的左上角编号位置（0，0））：" << endl;
    cin >> firstX >> firstY;
    //printT();
    visit(firstX, firstY, 1);
    cout << "Count " << countN << endl;
}
```

4.23

假设以块链结构作串。试编写判别给定串是否具有对称性的算法。

```
int LString_Palindrome(LString L)//判断以块链结构存储的串L是否为回文序列，是则返回1，否则返回0
{
    InitStack(S);
    p=S.head;i=0;k=1;
    for(k=1;k<=S.curlen+1;k++)
    {
        if(k<=S.curlen/2)Push(S,p->ch[i]);
        else if(k>(S.curlen+1)/2){
            Pop(S,c);
            if(p->ch[i]!=c)return 0;
        }
        if(++i==CHUNKSIZE)
        {
            p=p->next;
            i=0;
        }
    }
    return 1;
}
```

5.36

编写按上题描述的格式输出广义表的递归算法

```
void GList_PrintList(GList A)
{
    if(!A)printf("()");
    else if(!A->tag)printf("%d",A->atom);
    else
    {
        printf("(");
        for(p=A;p;p=->ptr.tp)
        {
            GList_PrintList(p->ptr.hp);
            if(p->ptr.tp)printf(",");
        }
        printf(")");
    }
}
```

6.53

试编写算法，求给定二叉树上从根节点到叶子节点的一条其路径长度等于树的深度减一的路径，若这样的路径存在多条，则输出路径终点（叶子结点）在“最左"的一条。

```
Status MaxPathBiTree(BiTree& T)
```

```
{
    if(T){
        if(BiTDepth(T)-BiTDepth(T->lchild)!=1)
            DelBiTree(T->lchild);
        else
            MaxPathBiTree(T->lchild);
        if(BiTDepth(T)-BiTDepth(T->rchild)!=1)
            DelBiTree(T->rchild);
        else
            MaxPathBiTree(T->rchild);
    }
    return OK;
}
// 从根到叶子最长路径中最左方的路径树
Status LMaxPathBiTree(BiTree& T)
{
    if(T){
        if(BiTDepth(T)-BiTDepth(T->lchild)==1){
            DelBiTree(T->rchild);
            LMaxPathBiTree(T->lchild);
        }
        else{
            DelBiTree(T->lchild);
            if(BiTDepth(T)-BiTDepth(T->rchild)==1)
                LMaxPathBiTree(T->rchild);
            else
                DelBiTree(T->rchild);
        }
    }
    return OK;
}
```

6.74

试写一递归算法，以6.73给定的树的广义表表示法的字符序列形式输出以孩子-兄弟链表表示的树。

```
void PrintGlist_CSTree(CSTree T)
{
    printf("%c",T->data);
    if(T->firstchild)
    {
        printf("(");
        for(p=T->firstchild;p;p=p->nextsib)
        {
            PrintGist_CSTree(p);
            if(p->nextsib)printf(",");
        }
        printf(")");
    }
}
```

7.24

试利用栈的基本操作编写，按深度优先搜索策略遍历一个强连通图的非递归形式的算法。算法中不规定的具体的存储结构，而将图Graph看成是一种抽象的数据类型，

```
void STraverse_Nonrecursive(Graph G)
{
    int visited[MAXSIZE];
    InitStack(S);
    Push(S,GetVex(S,1));
    visit(1);
    visited=1;
    while(!StackEmpty(S))
    {
        while(Gettop(S,i)&&i)
        {
            j=FirstADjVex(G,i);
            if(j&&!visited[j])
            {
                visit(j);
                visited[j]=1;
                Push(S,j);
            }
        }
        if(!StackEmpty(S))
        {
            Pop(S,j);
            Gettop(S,i);
            k=NextAdjVex(G,i,j);
            if(k&&!visited[k])
            {
                visit(k);
                visited[k]=1;
                Push(S,k);
            }
        }
    }
}
```

7.34

试编写一个算法，给有向无环图G中每个顶点赋以一个整数序号，并满足以下条件：若从顶点i至顶点j有一条弧，则应使i<j.

```
Status TopoSeq(ALGraph G,int new[])
{
    int indegree[MAXSIZE];
    FindIndegree(G,indegree);
    Initstack(S);
    for(i=0;i<G.vexnum;i++)
        if(!indegree[i])Push(S,i);
    count=0;
    while(!stackempty(S))
    {
```

```
        Pop(S,i);new[i]=++count;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {
            k=p->adjvex;
            if(!(--indegree[k]))Push(S,k);
        }
    }
    if(count<G.vexnum)return ERROR;
    return OK;
}
```