Homework 10.14 10.19

4.23 假设以块链结构作串的存储结构。试编写判别给定串是否具有对称性的算法,并要求算法的时间复杂度为O(*StrLength(S)*)。

类C实现:

```
Status LS_Symmetry(LString S){
    //借助栈完成以块链为存储结构的串的对称性判别,对称返回TRUE，反之FALSE。
    IntStack(T);
    p = S.head;
    i = 0;
    for(j = 1;j <= S.curlen;j++){
         if(j <= S.curlen / 2) //前半部分入栈
             Push(T, p -> ch[i]);
         else if(j > (S.curlen + 1) / 2) {//与后一半出栈比较
             Pop(T, e);
             if(p -> ch[i] != e)
                 return FALSE;
         }
         if(++i == CHUNKSIZE){//下一个块
             p = p->next;
             i = 0;
         }
     }
    return TRUE;
}//LS_Symmetry
```

5.7 设有三对角矩阵$(a_{ij})_{nxn}$将其三条对角线上的元素逐行地存于数组B[3n-2]中，使得B[k]=a_{ij}，求:
(1) 用i，j表示k的下标变换公式;
(2) 用k表示i，j的下标变换公式。

i，j = 1,2,...,n，即a_{ij}从a_{11}开始，B[0,...,3n-1]。

k = 2(i - 1) + j -1 - 1 = 2i + j - 3;

i = [$\frac{k+1}{3}$] + 1，[·] := 向下取整;

j = k + 3 - 2i = k + 1 - 2 [$\frac{k+1}{3}$] 。

5.27 试按教科书5.3.2节中定义的十字链表存储表示编写将稀疏矩阵B加到稀疏矩阵A上的算法。

类C实现:

```
Status AddSMatrix(CrossList &A,CrossList B){
    //十字链表表示稀疏矩阵的加法
    j = 1,flag = 0,tag;//flag判断重复位置的个数
    OLink p,q1,q2;
    if(A.mu != B.mu || A.nu != B.nu)    return ERROR;//行列数相同得以相加
    for(i = 1;i <= B.tu;i++){
        tag = 0;
        p = (OLNode*)malloc(sizeof(OLNode));
```

```
        if(!p)  exit(OVERFLOW);
        while(j <= B.nu){//B中找元素
            if(B.chead[j] != NULL){
                p->i = B.chead[j]->i;
                p->j = B.chead[j]->j;
                p->e = B.chead[j]->e;
                B.chead[j] = B.chead[j]->down;
                break;
            }
            else j++;
        }
        q1 = A.chead[p->j];     q2 = A.rhead[p->i];
        if(A.rhead[p->i] == NULL || p->j < A.rhead[p->i]->j){
            p->right = A.rhead[p->i];
            A.rhead[p->i] = p;
        }
        else{
            while(q2->right && p->j > q2->right->j)
                q2 = q2->right;
            if(q2->right){
                if(p->j == q2->right->j){
                    if(!tag){//不要重复累加值
                        q2->right->e += p->e ;
                        flag++;
                        tag = 1;
                    }
                }
                else{
                    p->right = q2->right;
                    q2->right = p;
                }
            }
            else    q2->right = p;
        }
        if(A.chead[p->j] == NULL || p->i < A.chead[p->j]->i){
            p->down = A.chead[p->j];
            A.chead[p->j] = p;
        }
        else{
            while(q1->down && p->i > q1->down->i)
                q1 = q1->down;
            if(q1->down){
                if(p->i == q1->down->i){
                    if(!tag){
                        q1->down->e += p->e ;
                        flag++;
                        tag = 1;
                    }
                }
                else{
                    p->down = q1->down;
                    q1->down = p;
                }
            }
            else    q1->down = p;
        }s
    }
    A.tu = A.tu + B.tu - flag;//更新个数
```

```
    return OK;
}
```

完整C实现:

```c
#include <stdio.h>
#include <stdlib.h>
#define OK      1
#define ERROR   0
#define OVERFLOW -2

typedef int Status;
typedef struct OLNode{
    int i,j;    // 该非零元的行和列下标
    int e;    // 非零元素值
    struct OLNode *right,*down; // 该非零元所在行表和列表的后继链域
}OLNode, *OLink;

typedef struct{// 行和列链表头指针向量基址,由CreatSMatrix_OL()分配
    OLink *rhead, *chead;
    int mu, nu, tu;         // 稀疏矩阵的行数、列数和非零元个数
}CrossList;

Status InitSMatrix(CrossList &M){
    M.rhead = M.chead = NULL;
    M.mu = M.nu = M.tu = 0;
    return OK;
}

Status CreateSMatrix(CrossList &M){...}//与书P104一致

Status PrintSMatrix(CrossList M){// 按行或按列输出稀疏矩阵M
    int i,j;
    OLink p;
    printf("%d行%d列%d个非零元素\n",M.mu,M.nu,M.tu);
    printf("请输入选择(1.按行输出 2.按列输出): ");
    scanf("%d",&i);
    switch(i){
    case 1:
        for(j = 1;j <= M.mu;j++){
            p = M.rhead[j];
            while(p){
                printf("%d行%d列值为%d\n",p->i,p->j,p->e);
                p = p->right;
            }
        }
        break;
    case 2:
        for(j = 1;j <= M.nu;j++){
            p = M.chead[j];
            while(p) {
                printf("%d行%d列值为%d\n",p->i,p->j,p->e);
                p = p->down;
            }
        }
    }
    return OK;
```

```
}

Status AddSMatrix(CrossList &A,CrossList B){//加法
    int j = 1,i,flag = 0,tag;
    OLink p,q1,q2;
    if(A.mu != B.mu || A.nu != B.nu)    return ERROR;
    for(i = 1;i <= B.tu;i++){
        tag = 0;
        p = (OLNode*)malloc(sizeof(OLNode));
        if(!p)  exit(OVERFLOW);
        while(j <= B.nu){
            if(B.chead[j] != NULL){
                p->i = B.chead[j]->i;
                p->j = B.chead[j]->j;
                p->e = B.chead[j]->e;
                B.chead[j] = B.chead[j]->down;
                break;
            }
            else j++;
        }
        q1 = A.chead[p->j];     q2 = A.rhead[p->i];
        if(A.rhead[p->i] == NULL || p->j < A.rhead[p->i]->j){
            p->right = A.rhead[p->i];
            A.rhead[p->i] = p;
        }
        else{
            while(q2->right && p->j > q2->right->j)
                q2 = q2->right;
            if(q2->right){
                if(p->j == q2->right->j){
                    if(!tag){//不要重复累加值
                        q2->right->e += p->e ;
                        flag++;
                        tag = 1;
                    }
                }
                else{
                    p->right = q2->right;
                    q2->right = p;
                }
            }
            else    q2->right = p;
        }
        if(A.chead[p->j] == NULL || p->i < A.chead[p->j]->i){
            p->down = A.chead[p->j];
            A.chead[p->j] = p;
        }
        else{
            while(q1->down && p->i > q1->down->i)
                q1 = q1->down;
            if(q1->down){
                if(p->i == q1->down->i){
                    if(!tag){
                        q1->down->e += p->e ;
                        flag++;
                        tag = 1;
                    }
                }
```

```
                else{
                    p->down = q1->down;
                    q1->down = p;
                }
            }
            else    q1->down = p;
        }
    }
    A.tu = A.tu + B.tu - flag;
    return OK;
}

int main(){
    CrossList A,B;
    InitSMatrix(A);
    InitSMatrix(B);
    printf("创建矩阵A: ");
    CreateSMatrix(A);
    PrintSMatrix(A);
    printf("创建矩阵B:(与矩阵A的行、列数相同，行、列分别为%d,%d)\n",A.mu,A.nu);

    CreateSMatrix(B);
    PrintSMatrix(B);
    printf("矩阵A+B: ");
    AddSMatrix(A,B);
    PrintSMatrix(A);
    return 0;
}
```

结果测试:

```
创建矩阵A: 请输入稀疏矩阵的行数 列数 非零元个数:(space) 3 3 3
请按任意次序输入3个非零元的行 列 元素值:(空格)
1 2 3
3 3 3
2 3 4
3行3列3个非零元素
请输入选择(1.按行输出 2.按列输出): 1
1行2列值为3
2行3列值为4
3行3列值为3
创建矩阵B:(与矩阵A的行、列数相同，行、列分别为3,3)
请输入稀疏矩阵的行数 列数 非零元个数:(space) 3 3 4
请按任意次序输入4个非零元的行 列 元素值:(空格)
1 2 5
3 2 1
1 1 1
2 1 7
3行3列4个非零元素
请输入选择(1.按行输出 2.按列输出): 1
1行1列值为1
1行2列值为5
2行1列值为7
3行2列值为1
矩阵A+B: 3行3列6个非零元素
请输入选择(1.按行输出 2.按列输出): 1
1行1列值为1
1行2列值为8
2行1列值为7
2行3列值为4
3行2列值为1
3行3列值为3
```

时空分析:

*CreateSMatrix*的时间复杂度为O(s×t)，空间复杂度为O(m+n+t+2)，其中s=*max*{m,n}，t为非零元个数，m、n为行列数。

*AddSMatrix*的时间复杂度为O(s_B×t_B)，空间复杂度为O(t_B)，其中s_B=*max*{m_A,n_A,n_B}，t_B为B中非零元个数，m_A、n_A为A的行列数，n_B是B的列数。

2020/10/21 12:59