HOMEWORK 11.30 12.2

PB19010450 和泳毅

7.31 试完成求有向图的强连通分量的算法,并分析算法的时间复杂度。

```
int visited[MAX_VERTEX_NUM];//访问标志
int finished[MAX_VERTEX_NUM];//存储DFS访问到的点
Status SCC(OLGraph G) {
    count = 0;
    DFSTraverse(G, Elem);
    InverseGraph(&G);
    for(i = 0;i < G.vexnum;i++)
        visited[i] = FALSE;
    for(i = 0;i < count;i++){
        v = LocateVex(G, finished[i]);
        if(visited[v])  continue;//已访问
        DFS_Inverse(G, v);
        printf("\n");
    }
    return OK;
}

Status InverseGraph(OLGraph &G){
    //十字链表
    H = G;
    for(i = 0; i < H.vexnum; i++){
        H.xlist[ i ].firstin = NULL;
        H.xlist[ i ].firstout = NULL;
    }
    for(i = 0;i < G.vexnum;i++){
        while(G.xlist[ i ].firstin != NULL){
            t = G.xlist[ i ].firstin;
            G.xlist[ i ].firstin = t->hlink;
            t->headvex <-> t->tailvex;//逆置弧的方向
            t->hlink = NULL;    t->tlink = NULL;
            q = H.xlist[ t->headvex ].firstin;
            if(!q)  H.xlist[ t->headvex ].firstin = t;
            else{
                if(t->tailvex < q->tailvex){
                    t->hlink = H.xlist[ t->headvex ].firstin;
                    H.xlist[ t->headvex ].firstin = t;
                }
                else if(t->tailvex > q->tailvex){
                    p = q->hlink;
                    while(p != NULL && t->tailvex > p->tailvex){
                        q = p;  p = p->hlink;
                    }
                    if(p == NULL || t->tailvex < p->tailvex){
                        t->hlink = p;   q->hlink = t;
                    }
                }
                else {
```

```
                    // 等于时不处理
                }
            }
            q = H.xlist[ t->tailvex ].firstout;
            if(!q)  H.xlist[ t->tailvex ].firstout = t;
            else{
                if(t->headvex < q->headvex){
                    t->tlink = H.xlist[ t->tailvex ].firstout;
                    H.xlist[ t->tailvex ].firstout = t;
                }
                else if(t->headvex > q->headvex){
                    p = q->tlink;
                    while(p != NULL && t->headvex > p->headvex){
                        q = p;  p = p->tlink;
                    }
                    if(p == NULL || t->headvex < p->headvex){
                        t->tlink = p;   q->tlink = t;
                    }
                }
                else {
                    // 等于时不处理
                }
            }
        }
    }
    G = H;
    return OK;
}

Status Elem(VertexType e){
    finished[count++] = e;
    return OK;
}

void DFS_Inverse(OLGraph G, int v){
    if(visited[v] == TRUE)  return;
    visited[v] = TRUE;
    printf("%c ",GetVex(G,v));
    for(p = FirstAdjVex(G, G.xlist[v].data);
        p >= 0;p = NextAdjVex(G, G.xlist[v].data, G.xlist[p].data))
        DFS_Inverse(G,p);  // 对尚未访问的顶点调用DFS
}

//SCC:  把调用的函数看做基本操作, T=O(n);
//DFS_Inverse:  T=O(n);
//InverseGraph: T=O(n^2).
```

7.26 试证明,对有向图中顶点适当地编号,可使其邻接矩阵为下三角形且主对角线为全零的**充要条件**是:该有向图不含回路。然后写一算法对无环有向图的顶点重新编号,使其邻接矩阵变为下三角形,并输出新旧编号对照表。

证明:

充分性:弧尾顶点的编号 > 弧头顶点的编号 => 在邻接矩阵中,非零元素属于下三角矩阵

必要性:要使上三角为 0 ,则不允许出现弧头顶点编号 > 弧尾顶点编号的弧,否则出现回路

严格证明：

必要性：
假设有向图G中 $\exists_{回路}\ v_a->v_b->v_c->...->v_a$

对a、b、c...调整顺序后，该回路可以被表示为$v_n->v_{n-1}->v_{n-2}->...->v_1->v_n$
则一定 $\exists$ 边$v_i->v_j$，$i<j$
而$i<j$的边是无法保存到邻接矩阵的下三角，矛盾！

充分性：
假设顶点集V(G)无论如何调整顺序，邻接矩阵的上三角至少包含一条边的信息
则至少 $\exists$ 一条边为$v_i->v_j$，$i<j$
由对称性，$v_i->v_j$可以调整为$v_j->v_i$，如果无法调整，则下三角区已 $\exists\ v_j->v_i$
$\exists\ v_i->v_j$,$v_j->v_i$ $=>$ $\exists$ 回路，矛盾!
综上：对有向图中顶点适当地编号,可使其邻接矩阵为下三角形且主对角线为全零 $<=>$ 该有向图不含回路。

```
Status TopologicalSort(ALGraph G){
    FindInDegree(G,indegree);
    InitStack(&S);
    for(i = 0;i < G.vexnum;i++)//建零入度顶点栈
        if(indegree[i] == 0)    Push(S,i);
    count = 0;//对输出顶点计数
    while(!StackEmpty(S)){
        Pop(S,i);
        temp[count++] = i;
        for(p = G.vertices[i].firstarc; p != NULL; p = p->nextarc){
            k = p->adjvex;
            if(!(--indegree[k]))    Push(S, k);
        }
    }
    if(count < G.vexnum){
        return ERROR;
        printf("有向图有回路\n");
    }
    else{
        printf("旧编号序列:");
        for (i = 1;i <= G. vexnum; i++)
            printf("%c", G. vexs[temp[i]]);
        printf("\n");
        printf("新编号序列:");
        for(i = G. vexnum;i >= 1; i--)
            printf("%c", G.vexs[temp[i]]);
        printf("\n");
        return OK;
    }
}

void FindInDegree(ALGraph G, int indegree[MAX_VERTEX_NUM]) {
    for(i = 0; i < G.vexnum; i++)   indegree[i] = 0;
    for(i = 0; i < G.vexnum; i++){
        p = G.vertices[i].firstarc;
        while(p != NULL){
            indegree[p->adjvex]++;
            p = p->nextarc;
        }
    }
```

```
}
```

7.35 若在DAG图中存在一个顶点r,在r和图中所有其他顶点之间均存在由r出发的有向路径,则称该DAG图有根。试编写求DAG图的根的算法。

```
int root = -1;//存根
int indegee[MAX_VERTEX + 1];//入度
Status DAG_Root(ALGraph G){
//DAG图至多1个root，若存在返回TRUE，根存于root，此时root不为-1，反之返回FALSE。
    for(i = 0;i < G.vexnum;i++)
        indegree[i] = 0;//入度为0
    for(i = 0;i < G.vexnum;i++){
        p = G.vertices[i].firstarc;//该顶点首个邻接点
        while(p != NULL){
            indegree[p->adjvex]++;
            p = p->nextarc;
        }
    }
    flag = 0;
    for(i = 0;i < G.vexnum;i++)
        if(indegree[i] == 0){
            flag++;//根数
            root = i;//存根
        }
    if(flag == 1)   return TRUE;
    else    return FALSE;
}
```