Homework 11.23 11.25

PB19010450 和泳毅

7.29 试写一个算法,在以**邻接矩阵**方式存储的有向图G中求顶点 i 到顶点 j 的不含回路的、长度为 k 的路径数。

类C描述:

```
int count = 0;
int visited[MAX_VERTEX_NUM];//访问标记

int SimplePath(MGraph G,int i,int j,int k){
    for(i = 1;i <= MAX_VERTEX_NUM;i++)  visited[i] = FALSE;
    DFS_7_29(G,i,j,k);
    return count;
}

void DFS_7_29(MGraph G,int i,int j,int k){
    visited[i] = TRUE;//已访问
    Push(S,i);
    if(i == j){
        for(p = S.base,n = -1;p < S.top;p++,n++);
        if(n == k) count++;
        Pop(S,e);
        visited[i] = FALSE;
        return;
    }
    for(q = FirstAdjVex(G,G.vexs[i]);q;
                        q = NextAdjVex(G,G.Vex[i],G.Vex[q])){
        if(!visited[q]) DFS_7_29(G,q,j,k);
    }
    if(!q){
        Pop(S,e);
        visited[i] = FALSE;
    }
}
```

7.30 试写一个求有向图G中所有简单回路的算法。

类C描述:

```
int visited[MAX_VERTEX_NUM]; //访问标记
int p[MAX_VERTEX_NUM]; //暂存路径
int k;
int path[MAX_VERTEX_NUM + 1][MAX_VERTEX_NUM + 1]; //存储找到的路径
    //[0][0] 存放路径的总个数   [1][0] 存放第一条路径的长度 [1][1]开始存放第一条的结点

Status AllSimpleCycle(MGraph G) {
```

```
    for(i = 0;i < MAX_VERTEX_NUM;i++)// path初始化
        for(j = 0;j < MAX_VERTEX_NUM;j++)
            path[i][j] = 0;
    for(i = 0;i <= G.vexnum;i++)    visited[i] = FALSE;
    for(i = 0;i <= G.vexnum;i++){
        k = 1; //路径的第一个结点
        p[k] = i; //第一个结点的编号为i
        DFS_7_30(G,i);
    }
    return OK;
}

void DFS_7_30(MGraph G,int i) {
    visited[i] = TRUE; //已访问
    for(u = 0;u < G.vexnum;u++){    //用u遍历i的邻边
        if(G.arcs[i][u].adj == 0) continue; //i到u没有边：退出
        if(u > p[1] && visited[u] == FALSE) {
            p[++k] = u; //记到路径上
            DFS_7_30(G,u); //往u继续走
        }
        if(u == p[1]) {
            path[0][0]++; //总路径的个数
            path[ path[0][0] ][0] = k; // 路径长度
            for(j = 1;j <= k;j++) //将求得的路径存入路径数组
                path[ path[0][0] ][j] = p[j]; //回路的终点即起点，不存入数组
        }
    }
    visited[ p[k] ] = 0; //返回上一个顶点
    k--;
}
```

7.27　采用**邻接表**存储结构,编写一个判别无向图中任意给定的两个顶点之间是否存在一条长度为k的简单路径的算法。

类C描述：

```
int visited[MAX_VERTEX_NUM];
int node = 0;//第几个结点
Status ExistSimplePath(ALGraph G,int v,int w,int k) {
    visited[v] = ++node;//第几次访问
    if(v == w && k == 0) return TRUE;//存在
    else if(k > 0) {//从v的邻边开始找
        for(p = G.vertices[v].firstarc;p;p = p->next){
            if(visited[p->adjvex] == 0){//没访问过
                if(ExistSimplePath(G,p->adjvex,w,k - 1))
                    return TRUE; //从邻边p中找到了
                else    visited[p->adjvex] = 0;//从p->adj退出
                node--; //退一步继续
            }
            else    continue;//已访问过
        }
    }
    return FALSE;
}
```

7.32 试修改**普里姆算法**，使之能在**邻接表**存储结构上实现求图的最小生成森林，并分析其时间复杂度(森林的存储结构为**孩子-兄弟链表**)。

类C描述：

```
typedef struct {
    VertexType adjvex;      // 顶点子集U中的顶点
    int lowcost;            // 顶点子集V-U到当前顶点adjvex的边的权值
} closedge[MAX_VERTEX_NUM]; // 辅助数组

CSTree MinSpanTree_PRIM(ALGraph G){
    T = NULL;
    curRoot = NULL;                   // 追踪森林中子树的根结点
    parent[MAX_VERTEX_NUM] = {NULL}; // 追踪各顶点在森林中的位置
    for(j = 0;j < G.vexnum;j++)
        closedge[j].lowcost = INT_MAX;
    for(i = 0;i < G.vexnum;i++){
        // 从顶点子集V-U中选出下一个候选顶点以便后续加入到最小生成树
        k = minimum(G);
        r = (CSTree)malloc(sizeof(CSNode));
        r->firstchild = r->nextsibling = NULL;
        if(k == -1) {// 确定森林中子树的根
            for(k = 0;k < G.vexnum;k++){
                if(closedge[k].lowcost == INT_MAX){
                    r->data = G.vertices[k].data;
                    parent[k] = r;
                    if(curRoot == NULL) T = curRoot = r;
                    else{// 链接到森林中下一棵树
                        curRoot->nextsibling = r;
                        curRoot = r;
                    }
                    break;
                }
            }
        }
        else{
            r->data = G.vertices[k].data;// 获取父结点在图中的序号
            j = LocateVex(G,closedge[k].adjvex);
            if(parent[j]->firstchild == NULL)
                parent[j]->firstchild = r;
            else{//头插法
                r->nextsibling = parent[j]->firstchild->nextsibling;
                parent[j]->firstchild->nextsibling = r;
            }
            parent[k] = r;
            printf("%c --%2d--
%c\n",closedge[k].adjvex,closedge[k].lowcost,G.vertices[k].data);
        }
        closedge[k].lowcost = 0;
        // 新顶点进入顶点子集U后，需要更新顶点子集U与顶点子集V-U的边的信息
        for(p = G.vertices[k].firstarc;p != NULL;p = p->nextarc){
            j = p->adjvex;
            if(p->info->weight < closedge[j].lowcost){
                closedge[j].adjvex = G.vertices[k].data;
```

```
                closedge[j].lowcost = p->info->weight;
            }
        }
    }
    return T;
}

int minimum(ALGraph G){
    k = -1;
    min = INT_MAX;
    for(i = 0;i < G.vexnum;i++){
        if(closedge[i].lowcost != 0 && closedge[i].lowcost < min){
            min = closedge[i].lowcost;
            k = i;
        }
    }
    return k;
}

```