```
typedef struct  // 图的定义
{
    int edges[maxsize][maxsize];     // 邻接矩阵定义，如果是有权图，则在词句中将int改为float
    int n, e;    // 分别为顶点数和边数
    int vex[maxsize];     // 存放结点信息
}MGraph;

typedef struct ArcNode
{
    int adjvex;                 // 改变所指向的结点的位置
    struct ArcNode *nextarc;    // 指向下一条边的指针
    int info;                   // 该边的相关信息(如权值) 若无要求 可以不写
}ArcNode;

typedef struct
{
    char data;       // 顶点信息
    ArcNode *firstArc;  // 指向第一条边的指针
}VNode;

typedef struct
{
    VNode adjlist[maxsize]; // 邻接表
    int n, e;    // 顶点数和边数
}AGraph;     // 图的邻接表类型

// 图的深度优先遍历
int visited[maxsize];
void DFS(AGraph *G, int v){
    ArcNode *p;
    visited[v] = 1;
    Visit(v);
    p = G->adjlist[v].firstArc;
    while(p){
        if(visited[p->adjvex] == 0)
            DFS(G, p->adjvex);
        p = p->nextarc;
    }
}
```

```
// 图的广度优先遍历
int visited[maxsize];
void BFS(AGraph *G, int v){
    ArcNode* p;
    int que[maxsize], front = 0, rear = 0;
    int j;
    Visit(v);
    visited[v] = 1;
    rear = (rear + 1) % maxsize;
    que[rear] = v;
    while (front != rear)
    {
        front = (front + 1) % maxsize;
        j = que[front];
        p = G->adjlist[j].firstArc;
        while(p){
            if(visited[p->adjvex] == 0){
                Visit(p->adjvex);
                visited[p->adjvex] = 1;
                rear = (rear + 1) % maxsize;
                que[rear] = p->adjvex;
            }
            p = p->nextarc;
        }
    }

}
```

```
// Dijkstra 某一顶点到其余个顶点的最短路径

void Dijkstra(MGraph g, int v, int dist[], int path[]){
    int set[maxsize];
    int min, i, j, u;
    /* 对各数组进行初始化*/
    for(i = 0; i < g.n; ++i){
        dist[i] = g.edges[v][i];
        set[i] = 0;
        if(g.edges[v][i] < __INT_MAX__)
            path[i] = v;
        else
            path[i] = -1;
    }
    set[v] = 1;
    path[v] = -1;
    /*初始化结束*/
    for(i = 0; i < g.n - 1; ++i){
        min = __INT_MAX__;
        /*这个循环每次从剩余顶点中选出一个顶点，通往这个顶点的路径在通往所有剩余顶点的路径中是长度最短的*/
        for(j = 0; j < g.n; ++j){
            if(set[j] == 0 && dist[j] < min){
                u = j;
                min = dist[j];
            }
            set[u] = 1; // 将选出的顶点并入最短路径中
            /*这个循环以刚并入的顶点作为中间顶点，对所有通往剩余顶点的路径进行检测*/
            for(j = 0; j < g.n; ++j){
                /* 这个if语句判断顶点u的加入是否会出现通往顶点j的更短的路径，如果出现，则改变原来路径及其长度，否则什么
                if(set[j] == 0 && dist[u] + g.edges[u][j] < dist[j]){
                    dist[j] = dist[u] + g.edges[u][j];
                    path[j] = u;
                }
            }
            /*dist[]数组中存放了v点到其余顶点的最短路径长度，path[]中存放v点到其余个顶点的最短路径*/
        }
    }
}
```

```
// Floyd 任意一对顶点间的最短路径
void Floyd(MGraph g, int Path[][maxsize]){
    int i, j, k;
    int A[maxsize][maxsize];
    /*这个双循环对数组A[][]和Path[][]进行了初始化*/
    for(i = 0; i < g.n; ++i){
        for(j = 0; j < g.n; ++j){
            A[i][j] = g.edges[i][j];
            Path[i][j] = -1;
        }
        /*下面这个三层循环是本算法的主要操作，完成了以k为中间点对所有的顶点对{i,j}进行检测和修改*/
        for(k = 0; k < g.n; ++k){
            for(i = 0; i < g.n; ++i){
                for(j = 0; j < g.n; ++j){
                    if(A[i][j] > A[i][k] + A[k][j]){
                        A[i][j] = A[i][k] + A[k][j];
                        Path[i][j] = k;
                    }
                }
            }
        }
    }
}
```