6.2.1图的遍历-DFS

深度优先搜索（ ）

点亮所有的灯

类似于树的先序遍历

若有 个顶点， 条边，时间复杂度是

1. 用邻接矩表存储图：
2. 用邻接矩阵存储图：
3. 实现简单，代码量少
4. 是用于解决连通性问题，可以很快找到一个可行解
5. 对于搜索树上的深层次结点，性能更好
6. 可能陷入无限循环（因为没有环路检测）
7. 对于搜索树上的广泛节点来说，可能会远离根结点，搜索效率低
8. 有可能找不到最优解，只会找到一个可行解

6.2.2图的遍历-BFS

广度优先搜索（ ）

若有 个顶点， 条边，时间复杂度是

1. 用邻接矩表存储图：
2. 用邻接矩阵存储图：
3. 能够找到最短路径，解决最短路径等问题
4. 对于搜索树上的广泛结点效率高，搜索途中若找到目标结点，该节点到根结点的路径就一定是最短路径
5. 不会陷入无限循环
6. 实现相对复杂，需要维护队列数据结构
7. 空间复杂度高，需要维护一个用于存储临时结点的队列，因此在空间受限的情况下可能不适用
8. 当目标结点位于深层次结点时，DFS的性能可能更好

6.2.3图的遍历-为什么需要两种遍历

6.2.4图的遍历-图不连通怎么办

1. **连通**：如果两个结点之间存在一条路径，就是连通
2. **路径**：是一系列结点的集合，其中任一对相邻的结点间都有图中的边。**路径长度**是路径中的边数（带权就是权重和）。如果两个结点之间的所有结点都不同，则称**简单路径**
3. **回路**：起点等于终点的路径
4. **连通图**：图中任意两结点均连通
5. **连通分量**：无向图的极大连通子图
    1. 极大顶点数：再加一个顶点就不连通了
    2. 极大边数：包含子图中所有顶点相连的所有边
    3. 这个概念比较tricky，要看一下PPT的示例
6. **强连通**：有向图两个顶点之间存在双向路径，就称他俩是强连通
7. **强连通图**：有向图中任意两顶点均强连通
8. **强连通分量**：有向图的极大强连通子图
    1. 这个概念也是比较tricky的

图不连通的话，意味着它有多个连通分量，调用一次DFS（或BFS）时，其实已经把该顶点所在的连通分量遍历了一遍

所以针对不连通的图，只要列出该图的所有连通分量即可

列出分量：只要该结点没有被访问过，就从它开始进行搜索

```c
/* 邻接表存储的图 - DFS */

void Visit( Vertex V )
{
    printf("正在访问顶点%d\n", V);
}

/* Visited[]为全局变量，已经初始化为false */
void DFS( LGraph Graph, Vertex V, void (*Visit)(Vertex) )
{   /* 以V为出发点对邻接表存储的图Graph进行DFS搜索 */
    PtrToAdjVNode W;

    Visit( V ); /* 访问第V个顶点 */
    Visited[V] = true; /* 标记V已访问 */

    for( W=Graph->G[V].FirstEdge; W; W=W->Next ) /* 对V的每个邻接点W->AdjV */
        if ( !Visited[W->AdjV] )    /* 若W->AdjV未被访问 */
            DFS( Graph, W->AdjV, Visit );    /* 则递归访问之 */
}



/* 邻接矩阵存储的图 - BFS */

/* IsEdge(Graph, V, W)检查<V, W>是否图Graph中的一条边，即W是否V的邻接点。   */
/* 此函数根据图的不同类型要做不同的实现，关键取决于对不存在的边的表示方法。*/
/* 例如对有权图, 如果不存在的边被初始化为INFINITY, 则函数实现如下:         */
bool IsEdge( MGraph Graph, Vertex V, Vertex W )
{
    return Graph->G[V][W]<INFINITY ? true : false;
}

/* Visited[]为全局变量，已经初始化为false */
void BFS ( MGraph Graph, Vertex S, void (*Visit)(Vertex) )
{   /* 以S为出发点对邻接矩阵存储的图Graph进行BFS搜索 */
    Queue Q;
    Vertex V, W;

    Q = CreateQueue( MaxSize ); /* 创建空队列, MaxSize为外部定义的常数 */
    /* 访问顶点S：此处可根据具体访问需要改写 */
    Visit( S );
    Visited[S] = true; /* 标记S已访问 */
    AddQ(Q, S); /* S入队列 */

    while ( !IsEmpty(Q) ) {
        V = DeleteQ(Q);  /* 弹出V */
        for( W=0; W<Graph->Nv; W++ ) /* 对图中的每个顶点W */
            /* 若W是V的邻接点并且未访问过 */
            if ( !Visited[W] && IsEdge(Graph, V, W) ) {
                /* 访问顶点W */
                Visit( W );
                Visited[W] = true; /* 标记W已访问 */
                AddQ(Q, W); /* W入队列 */
            }
    } /* while结束*/
}
```