复杂性分析

➢ 时间复杂度

- **DFS对每一条边处理一次（无向图的每条边从两个方向处理），每个顶点访问一次。**
- **采用邻接表表示时，有向图总代价为$\Theta(|V|+|E|)$，无向图为$\Theta(|V|+2|E|)$**
- **采用相邻矩阵表示时，处理所有的边需要$\Theta(|V|^2)$的时间，所以总代价为$\Theta(|V|+|V|^2)= \Theta(|V|^2)$。**

复杂度分析

- 广度优先搜索实质上与深度优先相同，只是访问顺序不同而已。

- 二者时间复杂度也相同

复杂性分析

➢ 拓扑排序的时间复杂度

- 采用相邻矩阵时，每次算法需要找所有入度为0的顶点，需要 $\Theta(|V|^2)$的时间，那么对|V|个顶点而言，总代价为$\Theta(|V|^3)$
- 当采用邻接表时，因为在顶点表的每个顶点中可以有一个字段来存储入度，所以只需$\Theta(|V|)$的时间，加上处理边、顶点的时间，总代价为$\Theta(2|V|+|E|)$

时间复杂性分析

- **如果不采用最小堆的方式，而是通过两两比较来扫描D数组**
  - 每次寻找权值最小结点，需要进行|V|次扫描，每次扫描|V|个顶点（$|V|^2$），而在更新D值处总共扫描|E|次
  - 总时间代价为$\Theta(|V|^2 + |E|) = \Theta(|V|^2)$
- **如果采用最小堆的方式**
  - 每次改变D[i].length，通过先删除再重新插入的方法来改变顶点i在堆中的位置
  - 或者仅为某个顶点添加一个新值(更小的)，作为堆中新元素(而不作删除旧值的操作，因为旧值被找到时，该顶点一定被标记为VISITED，从而被忽略)。
  - 不作删除旧值的缺点是，在最差情况下，它将使堆中元素数目由$\Theta(|V|)$增加到$\Theta(|E|)$，此时总的时间代价为$\Theta((|V|+|E|)\log|E|)$，因为处理每条边时都必须对堆进行一次重排

```
for(v=0;v<G.VerticesNum();v++)          //矩阵初始化，仅初始化邻接顶点
    for(Edge e=G.FirstEdge(v); G.IsEdge(e); e=G.NextEdge(e)){
            D[v][G.ToVertex(e)].length=G.Weight(e);
            D[v][G.ToVertex(e)].pre=v;
    }
//如果两顶点间最短路径经过顶点v，则有权值进行更新！
for(v=0;v<G.VerticesNum();v++)
      for(i=0;i<G.VerticesNum();i++)
            for(j=0;j<G.VerticesNum();j++)
                  if((D[i][v].length +D[v][j].length) < D[i][j].length){
                        D[i][j].length =D[i][v].length +D[v][j].length;
                        D[i][j].pre=D[v][j].pre;
                  }
}
```

➢ **时间复杂性：三重for循环，复杂度是O(n^3)，适合稠密图**

性能分析

- 使用了路径压缩，differ和UNION函数几乎是常数
- 假设可能对几乎所有边都判断过了，则最坏情况下算法时间代价为 O(|E|log |E|)，即堆排序的时间
- 适合于稀疏图