数据结构：

按逻辑：线性结构和非线性两类

线性：顺序和链式

非线性：树形和图

树形：树结构和二叉树

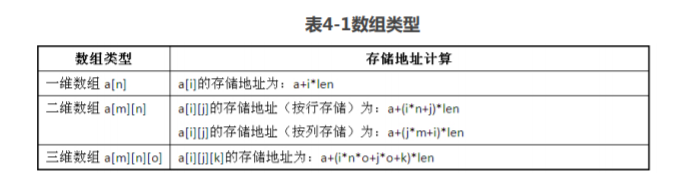
**线性表：**

顺序存储（数组）

优点：能随机存取线性表的任何一点

缺点：

1. 数组大小通常固定，不利于增加/减少节点个数
2. 插入和删除要移动其他节点，操作复杂



链式：

优点：每个节点的实际存储是任意的，利于增删

缺点：

1. 每个节点增加了一个后继指针成分，花费更多空间
2. 不便于随机访问任一节点

栈

队列

**稀疏矩阵**

如果一个矩阵的元素绝大部分为零，则称为稀疏矩阵

直接用一个二维数组表示稀疏矩阵，则会因存储太多的零元素而浪费大量的内存空间

用三元组数组和十字链表来存储

三元：

十字：

**字符串**

特点：  
\0作为结尾

KMP（解决字符串匹配问题）

前言：BF将目标串s第一个字符与找的字符串T第一个字符匹配，若同则再比较sT第二个字符，不同则比较s的第二和T的第一字符

KMP求next函数序列

Next[1]=0

Next[2]=1

其余求前缀和后缀相同字符串长度+1即是next值

**树**

条件：至少有一个节点（根节点），二叉树可以为空

名称：

节点的度：一个节点的子数数目（仅下一排）最大的为树的度

叶子节点：度为0

层次：根为1（自定义），其他层+1

存储结构

存储树需要把树的节点关系反映在存储结构中

标准存储

信息|树子节点指针

带逆存储结构

数据|树子节点指针|父指针

遍历

前序：先根，再左又  
后序：先左右，再根  
层次：先访问0层根节点，再左又访问1层节点~

**二叉树**

5种形态

1. 空
2. 一根
3. 一根一左一右
4. 一根一左
5. 一根一右

遍历：

前：根，左，右

中：左，根，右  
后：左，右，根

存储：

平衡二叉树

特点：树种任意节点的左右子树高度均相同

哈夫曼树：

**图**

有向图：若一个图中的每条边都是有方向的，则称为有向

无向图：若一个图中的每条边都是无方向的，则称为无向图。无向图的边是顶点的无序对，通 常使用（Vi，Vj）来表示一条边，无向图的边没有起点和终点，（Vi，Vj）和（Vj，Vi）表示的是同一条边

无向完全图：

顶点为n，可与其他n-1个点相连，则有n个顶点的无向图至多有n(n-1)/2 条边

有向完全图：

恰好有n(n-1)条边的有向图（一个点和其他点双向连接）

连通图：每两个点之间至少都有一条路径（没有点是没线连的）

特点：

存储：

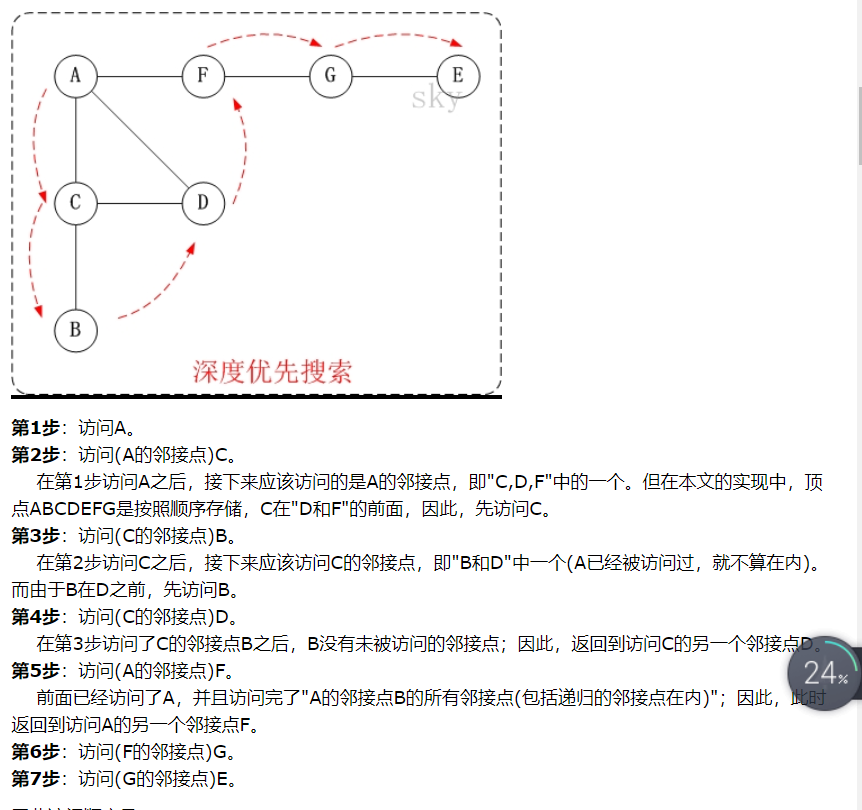
邻阶矩阵

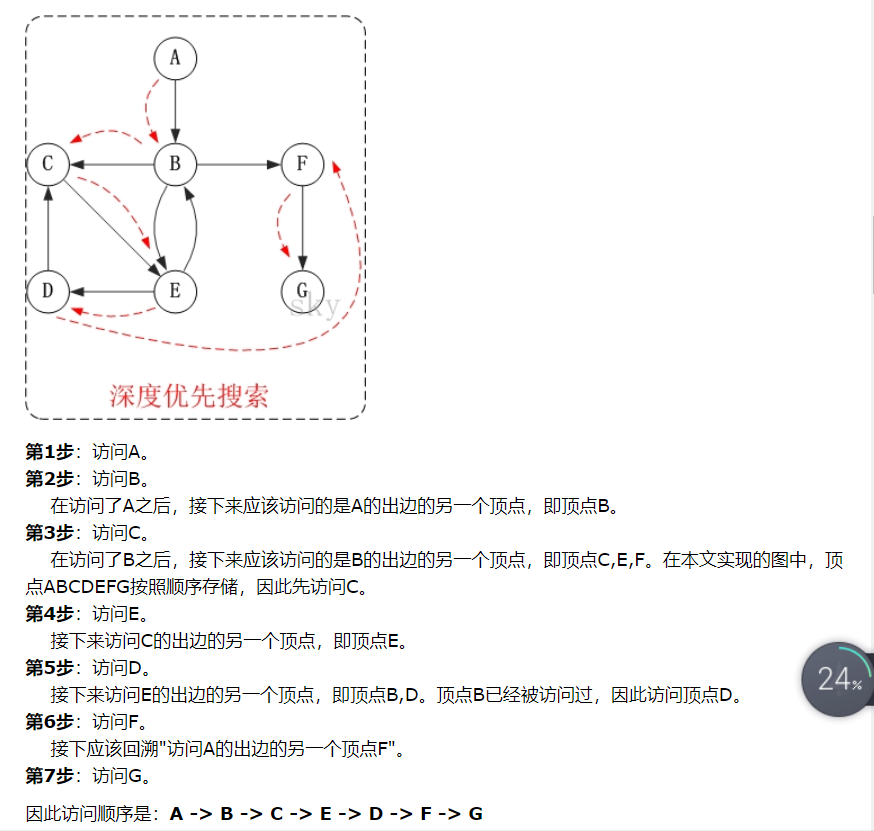
无向图：连接1，无连接0

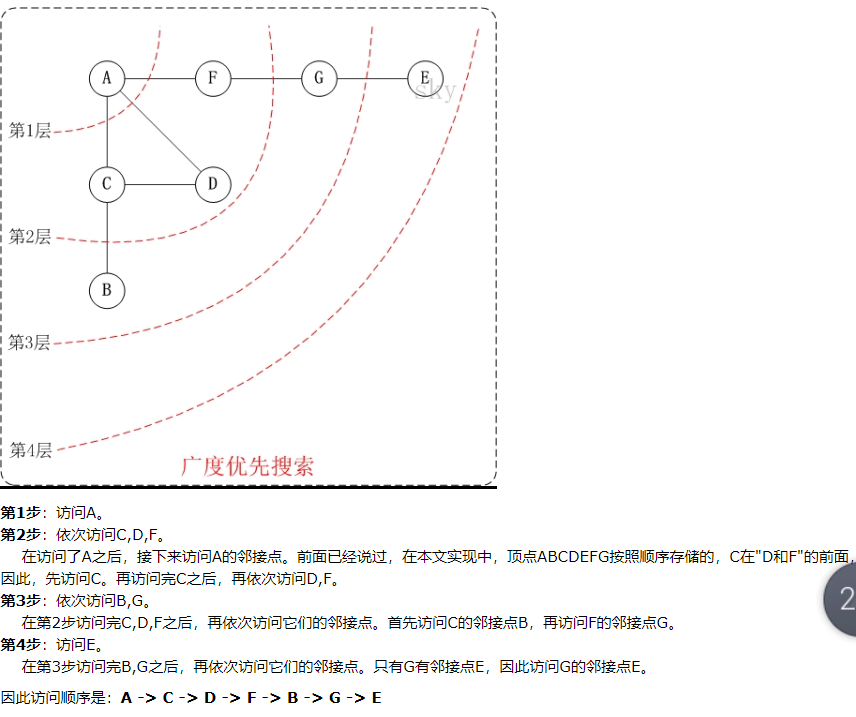
有：有箭头指向写距离，无就∞

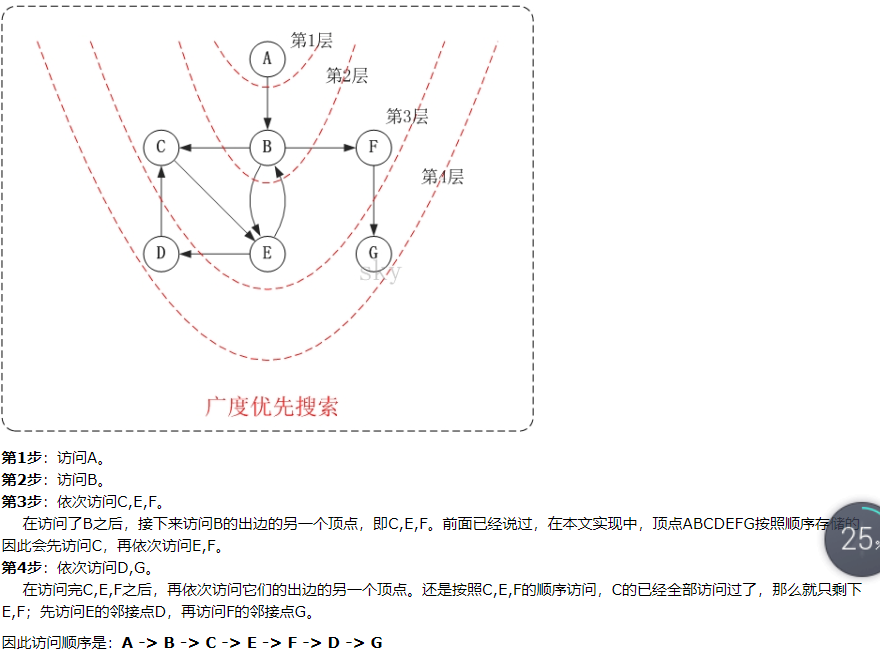
邻接表

遍历

或者按序号小-》大







最小生成树：

**插入排序**

直接插入排序

希尔排序

**选择排序**

直接选择排序

堆排序（树形选择排序）

**交换排序**

冒泡：

快速：

归并：

基数：

**查找：**

顺序查找

二分法查找：

分块查找：

**散列表**

冲突：两个不同的关键字，由于散列函数值相同而被映射到同一表位置上

冲突的频繁度与h（散列函数）和表的填满程度有关

解决：在散列表找一个空位

开放地址法：将所有节点放在散列表T中

拉链法：将互为同义词的结点链成一个单链表，头指针则放在散列表T中