1、程序设计的一般流程：

2、数据结构的基础知识

（1）数据结构是一门研究**非数值计算**的程序设计问题中，有关计算机的**操作对象**以及他们之间的**关系和操作**的学科。

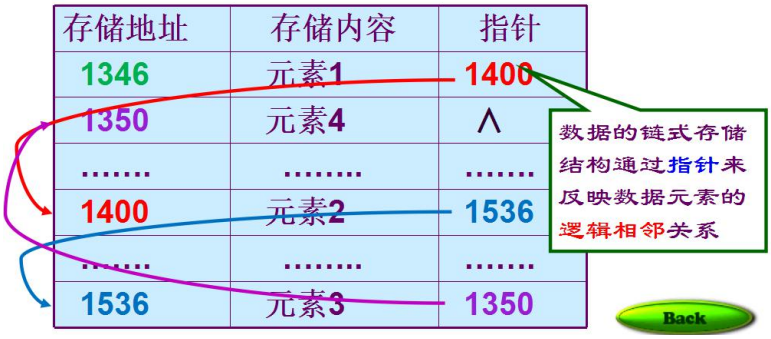
（2）**数据结构是相互之间存在一种或多种特定关系的数据元素的集合**，简单来说，数据结构是数据元素和数据元素关系的集合，即**数据结构={数据元素集合，元素间关系集合}**，一个数据元素可能由多个数据项组成。

（3）**数据的逻辑结构**是指数据元素之间的**逻辑关系**，分为：集合、线性结构（一对一）、树形结构（一对多）、图状结构（多对多），**面向问题**

（4）**数据的存储结构（物理结构）**是数据的逻辑结构在计算机存储器中的实现，分为**顺序存储结构**和**链式存储结构，面向计算机**

顺序存储结构：借助元素在存储器中的**相对位置**来表示数据元素间的逻辑相邻关系，不光逻辑上相邻，**存储位置也相邻**，因为会分配**连续的存储空间**

链式存储结构：借助指示元素存储地址的**指针**来表示数据元素间的**逻辑相邻**关系，只是逻辑上相邻，存储位置不一定相邻，比如下图就是链式存储结构：



（5）函数形参与实参的传递分为：值传递（数据的单向传递）、地址传递（数据的双向传递）、引用传递（数据的双向传递）

3、算法的基础知识

（1）算法是解决某一特定问题的**具体步骤的描述**，是指令的有限序列，特性有：有穷性、确定性、可行性、输入、输出

（2）时间频度：一个算法中的语句执行次数，记为 T(n)

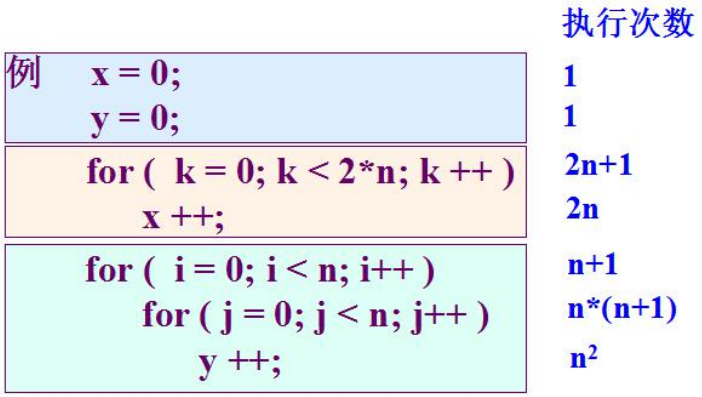
时间复杂度：基本操作重复执行次数的阶数（算法耗用时间的增长率），利用大 O 表示法 T(n)=O(f(n))，加法与乘法规则如下：

1、O(f(n))+O(g(n)) = max{O(f(n)), O(g(n)) }

2、O(f(cn)) = O(f(n)),　　c 是正整数

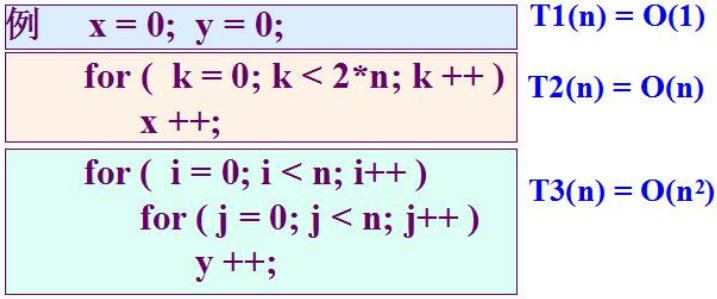
3、O(f(n))*O(g(n)) = O(f(n)*g(n))

具体计算实例如下：



时间耗费$T(n) = 4+ 6n+ 2n^2$　　$\lim_{n\to\infty} T(n)/n^2 = 2$

➢当n充分大时，T(n)与n^2在数量级上相同，记$T(n)=O(n^2)$



$T(n)= T1(n)+T2(n)+T3(n)= O(\max(1, n, n^2))= O(n^2)$

频度最大语句重复执行次数的阶数

常用时间复杂度有：

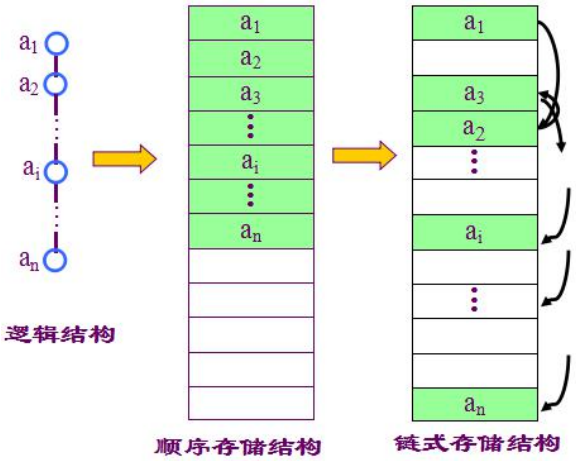
➢O(1)-----常量型

➢$O(n)$、$O(n^2)$、$O(n^3)$------多项式型

➢$O(\log_2 n)$、$O(n\log_2 n)$-------对数型

➢$O(2^n)$ 、$O(e^n)$------指数型

4、线性表介绍



（1）线性表的**顺序存储结构的定义**

用一组**地址连续的存储单元**存放线性表的数据元素，可以用一维数组或者**动态分配顺序存储单元**来实现。

（2）线性表的**链式存储结构的定义**

用一组**任意的存储单元**存放数据元素，利用**指针（链）**来实现用不相邻的存储单元存放逻辑上相邻的元素。每个数据元素除了存储**本身信息**外，还需要存储**其直接后继元素的地址**，所以每个结点都是由**数据域（元素本身信息）和指针域（直接后继元素的存储地址）**组成的。

（3）线性表的**顺序存储结构的优缺点**

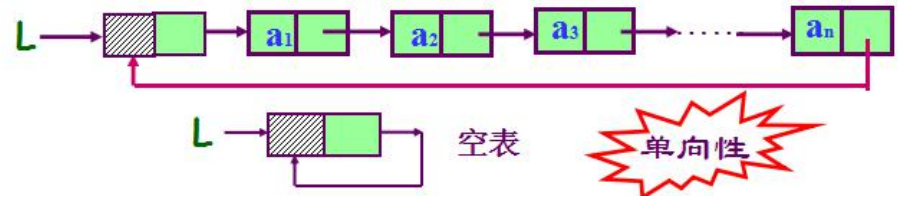
优点：逻辑相邻，物理地址也相邻； 可**随机存取**任一元素； 存储空间使用紧凑

缺点：插入、删除操作需要移动大量的元素，时间复杂度都是 T(n)=O(n)

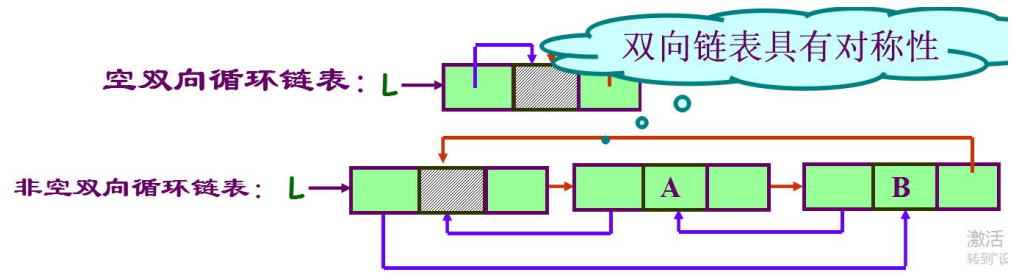
（4）线性表的**链式存储结构的分类**

①单链表：每个结点只含**一个指针域**用来指明后继元素，具有单向性

②循环列表：是单链表中一类特殊的链表，让表中**最后一个结点的指针指向头结点**，使链表构成**环状**，这样从表中任一结点出发均可找到表中其他结点，提高查找效率



③双向链表：每个结点含**两个指针域**用来指明前驱和后继元素，具有对称性



（5）**单链表存储结构的优缺点**

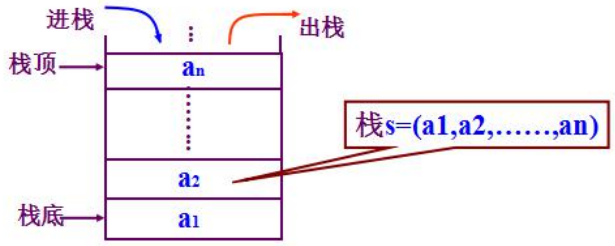
优点：是一种**动态结构**，整个存储空间可以被多个链表共用；**不需要预先分配**存储空间

缺点：**指针**占用**额外**存储空间； **不能随机存取**，查找速度慢； 单链表的表长是一个隐含值；在单链表的最后一个元素之后插入元素时需遍历整个链表，时间复杂度都是 T(n)=O(n)

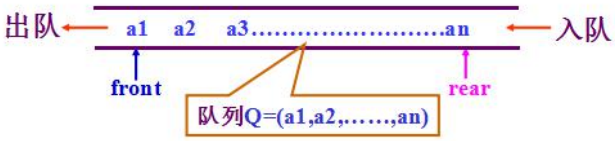
5、栈和队列

（1）是两种特殊的线性表，操作受限，限定插入和删除只能在表的“端点”进行

（2）**栈的定义**：限定仅在**表尾**进行**插入或删除**操作的线性表，特点是**后进先出**



（3）队列的定义：限定只能在表的一端（**队尾**）**进行插入**，在表的另一端（**队头**）**进行删除**的线性表，特点是**先进先出**



6、树及其存储结构

（1）基本术语

结点的度：结点拥有的子树数目

叶子：度为 0 的结点

树的度：一棵树中**最大的结点度数**

结点的层次：从根节点算起，根为第一层，依次往下为第二层....

树的深度：树中结点的**最大层次数**

（2）二叉树

定义：二叉树是 n（n≥0）个结点的有限集，或为空树，或由一个根节点和两颗分别称为左子树和右子树的互不相交的二叉树构成

特点：①每个结点至多有两颗子树（即不存在度大于 2 的结点）；②二叉树的子树有左右之分，且其次序不能任意颠倒

性质：

①在二叉树的**第 i 层上**至多有 2^{i-1} 个结点(i≥1)

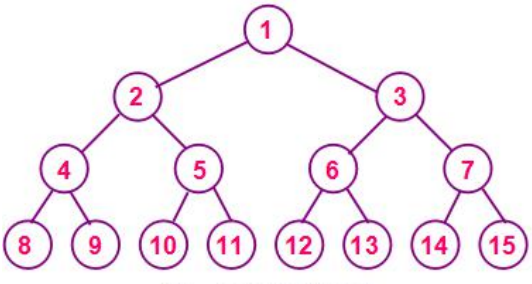
②**深度为 k** 的二叉树上总共至多含 2^k-1 个结点（k≥1）

③对任何一棵二叉树 T，如果其叶子结点数为 n_0，度为 2 的结点数为 n_2，则 $n_0=n_2+1$

（3）满二叉树

定义：一颗深度为 k 且总共有 2^k-1 个结点的二叉树

特点：每一层上的结点数都是最大结点数，**除最下层的叶子结点外，每个结点的度都为 2**

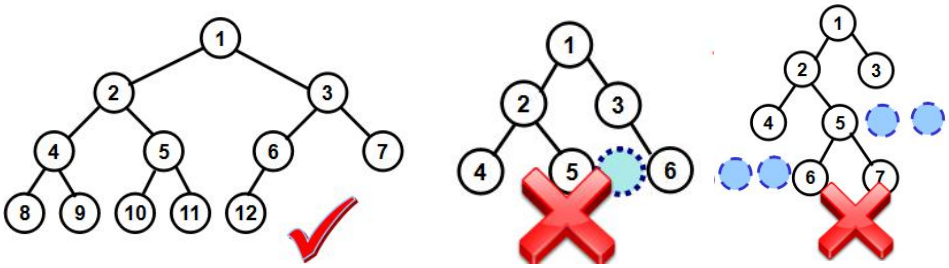


满二叉树及其编号

（4）完全二叉树

定义：深度为 k，有 n 个结点的二叉树当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时，称为完全二叉树

特点：①度小于 2 的结点只可能在层次最大的两层上出现，如果**某个结点没有左孩子**，那么**它一定没有右孩子**，该结点为叶子结点②**除最后一层外，其他每层都充满了结点**，最下面一层结点都集中在该层的**最左边**



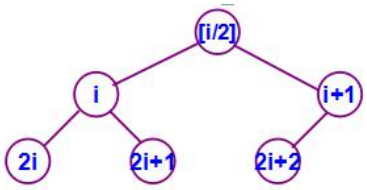
性质：①具有 n 个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor +1$（向下取整）

②如果对一棵有 n 个结点的完全二叉树的结点按层序编号，则对任一结点$i(1 \le i \le n)$，有：

如果 i=1，则结点 i 是二叉树的根，无双亲；如果 i>1，则其双亲是$\lfloor i/2 \rfloor$

如果 2i>n，则结点 i 无左孩子；如果 2i<=n，则其左孩子是 2i

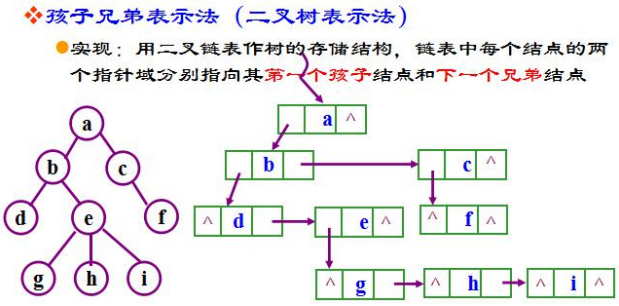
如果 2i+1>n，则结点 i 无右孩子；如果 2i+1<=n，则其右孩子是 2i+1



完全二叉树中结点i与双亲和左、右孩子关系

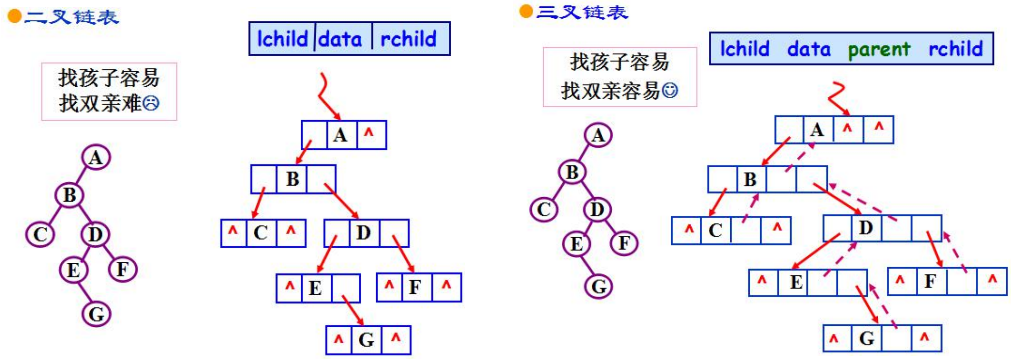
（5）普通树的存储结构

双亲表示法　　孩子表示法　　**孩子-兄弟表示法**



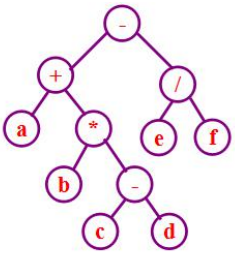
（6）二叉树的存储结构

顺序存储　　**二叉链表**　　三叉链表

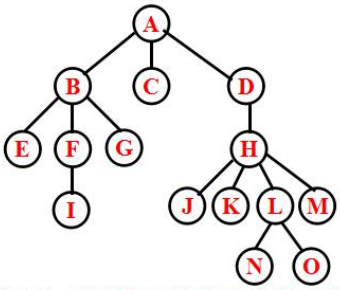


（7）二叉树的遍历方法（左下图）

先序遍历：先访问根节点，然后分别先序遍历左子树、右子树（**根、左、右**）
中序遍历：先中序遍历左子树，然后访问根节点，最后中序遍历右子树（**左、根、右**）
后序遍历：先后序遍历左、右子树，然后访问根节点（**左、右、根**）
按层次遍历：从上到下、从左到右访问各节点



先序遍历：- + a * b - c d / e f
中序遍历：a + b * c - d - e / f
后序遍历：a b c d - * + e f / -
层次遍历：-+/a*efb-cd

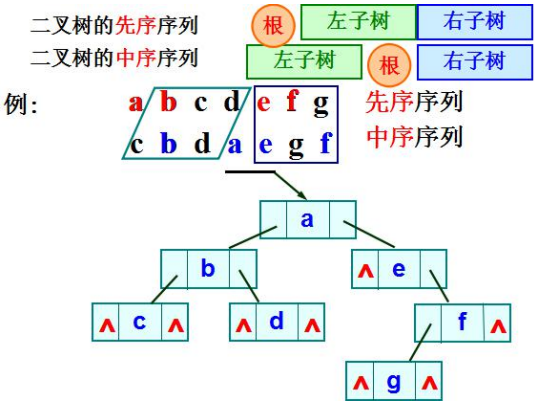


先序遍历：ABEFI GCDHJ KL NOM
后序遍历：EI FGBCJK NOL MHDA
层次遍历：ABCDE FGHI J KL MNO

（8）普通树（森林）的遍历方法（右上图）

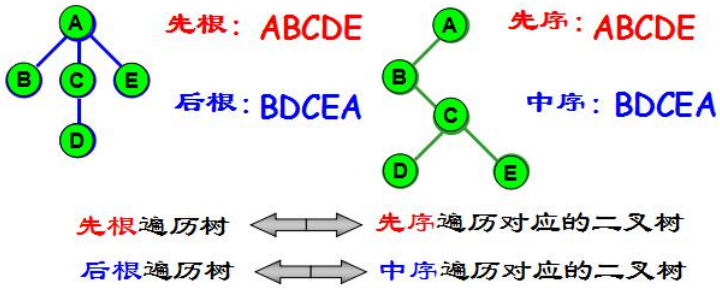
先根遍历：先访问根节点，然后依次先根遍历每棵子树（**根、左、右**）
后根遍历：先依次后根遍历每棵子树，然后访问根节点（**左、右、根**）
按层次遍历：从上到下、从左到右访问各节点

（9）由二叉树的先序序列和中序序列构造二叉树



注意：（**要构造二叉树，必须知道中序序列**）
已知二叉树**中序序列**和后序序列，**能**构造二叉树；
已知二叉树先序序列和后序序列，**不能**构造二叉树；
已知二叉树层次遍历序列和**中序序列**，**能**构造二叉树

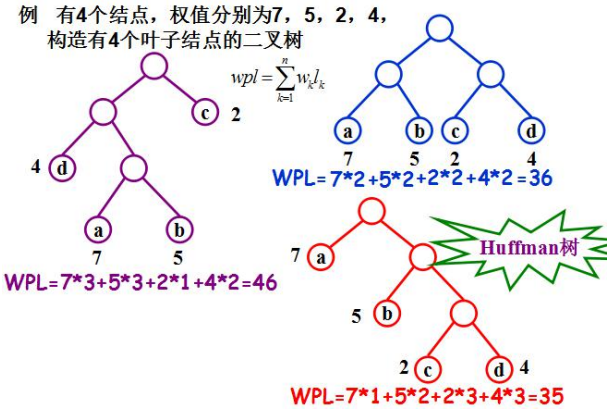
（10）普通树和二叉树的相互转化及遍历（采用孩子-兄弟二叉链表表示法）

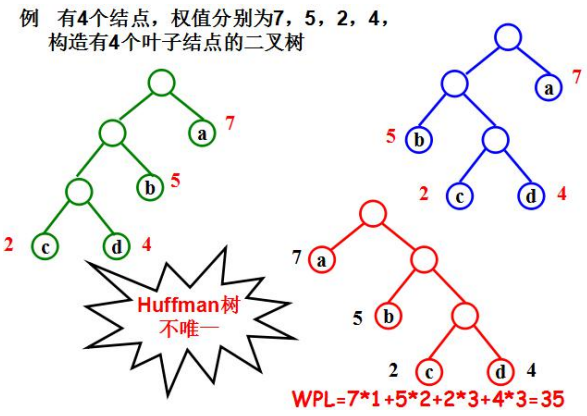


（11）哈夫曼树（Huffman 树）

设有 n 个权值{w1,w2,……wn}，构造多棵有 n 个叶子结点的二叉树，每个叶子的权值为 wi，则 **wpl 最小的二叉树叫哈夫曼树**，其中 wpl（Weighted Path Length）是带权路径长度，

记作：$wpl = \sum_{k=1}^{n} w_k l_k$
其中：w_k — 一权值
l_k — 一结点到根的路径长度





注意：Huffman 树的结构特点是权值越大的结点越靠近根节点，如左上图

Huffman 树不唯一，如右上图

7、图

（1）图的定义

❖图(Graph)——由两个集合V(G)和VR(G)组成,记为G=(V,VR)

其中：V(G)是顶点的非空有限集

VR(G)是边的有限集合，边是顶点的有序对或无序对

❖有向图(Digraph)——由两个集合V(G)和VR(G)组成

其中：V(G)是顶点的非空有限集

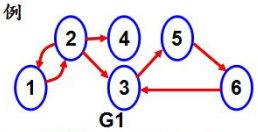
VR(G)是有向边（即弧）的有限集合，弧是顶点的有序对，记<V,W>，V为弧尾(Tail)，W为弧头(Head)

❖无向图(Undigraph)——由两个集合V(G)和VR(G)组成

其中：V(G)是顶点的非空有限集

VR(G)是边的有限集合，边是顶点的无序对，记(v,w)或(w,v)，并且(v,w)=(w,v)

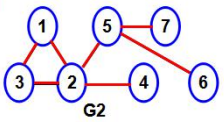
例



图G1中：V(G1)={ 1,2,3,4,5,6 }

E(G1)={<1,2>, <2,1>, <2,3>, <2,4>, <3,5>, <5,6>, <6,3>}

例



图G2中：V(G2)={1,2,3,4,5,6,7}

E(G2)={(1,2), (1,3), (2,3), (2,4),(2,5), (5,6), (5,7)}

（2）图相关的术语

❖有向完全图——n个顶点的有向图最大边数是n(n-1)

❖无向完全图——n个顶点的无向图最大边数是n(n-1)/2；

❖权——与图的边或弧相关的数

❖网——带权的图

❖子图——如果图G(V,VR)和图G'(V',VR'),满足：

●V'⊆V 并且 VR'⊆VR

则称G'为G的子图；

❖邻接点——顶点v和顶点w之间存在一条边，则称v和w互为邻接点；边(v,w)和顶点v和w相关联

❖顶点的度

●无向图中，顶点的度为与每个顶点相连的边数

●有向图中，顶点的度分成入度与出度

◆入度：以该顶点为头的弧的数目

◆出度：以该顶点为尾的弧的数目；

❖路径——路径是顶点的序列$V=\{V_{i0},V_{i1},\ldots\ldots V_{in}\}$，满足

$(V_{ij-1},V_{ij})\in VR$ 或 $<V_{ij-1},V_{ij}>\in VR,(1<j\leq n)$

❖路径长度——沿路径边的数目或沿路径各边权值之和

❖回路——第一个顶点和最后一个顶点相同的路径

❖简单路径——序列中顶点不重复出现的路径

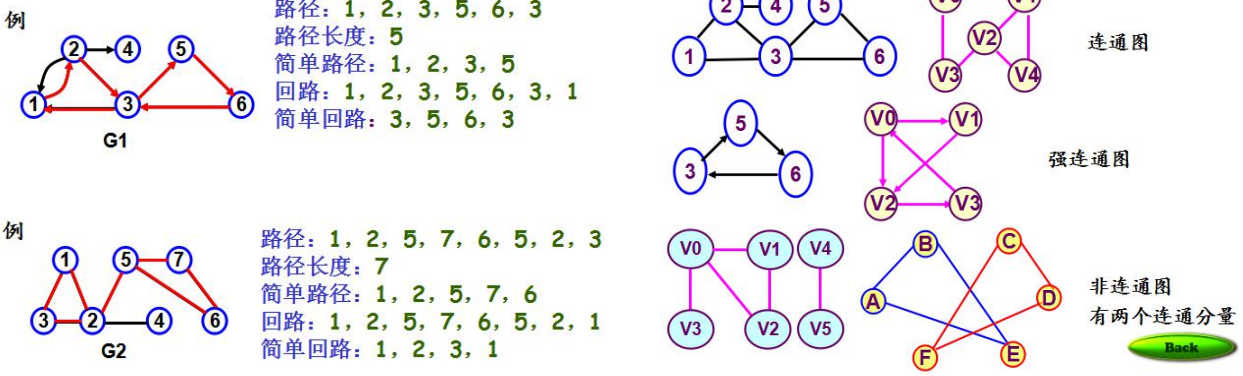
❖简单回路——除了第一个顶点和最后一个顶点外，其余顶点不重复出现的回路；

❖连通——从顶点V到顶点W有一条路径，则说V和W是连通的

❖连通图——图中任意两个顶点都是连通的图

❖连通分量——非连通图的每一个连通部分

❖强连通图——有向图中，如果对每一对$V_i,V_j\in V, V_i\neq V_j$,从$V_i$到$V_j$和从$V_j$到$V_i$都存在路径，则称G是~



（3）图的存储结构：邻接矩阵、邻接表

①邻接矩阵

定义：设 G=(V,E)是有 n>=1 个顶点的图，邻接矩阵存储结构用两个数组分别存储图中**顶点的信息**（一维数组）和**顶点间相关联的关系**（n 阶方阵），具体表示规则如下：

$$AdjMatrix[i][j]=\begin{cases}1 & (v_i,v_j)\text{ 或}<v_i,v_j>\in VR\\ 0 & \text{否则}\end{cases}$$ （不带权的图）

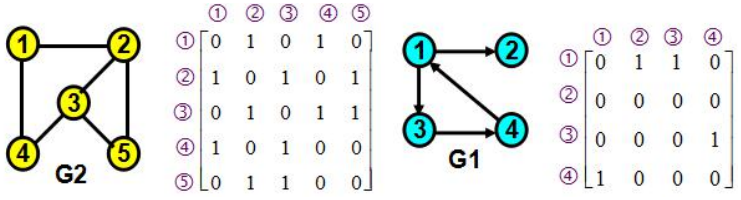
$$AdjMatrix[i][j]=\begin{cases}\omega_{ij} & (v_i,v_j)\text{ 或}<v_i,v_j>\in VR\\ 0\text{或}\infty & \text{否则}\end{cases}$$ （带权图）

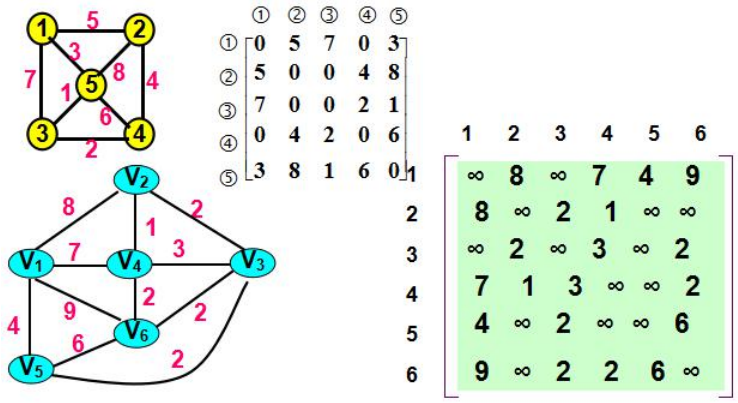
特点：无向图的邻接矩阵对称，可压缩存储；有向图的邻接矩阵不一定对称

无向图中顶点 Vi 的度是邻接矩阵中第 i 行元素之和；

有向图中顶点 Vi 的出度是邻接矩阵中第 i 行元素之和，入度是第 i 列元素之和

具体实例如下：





②邻接表

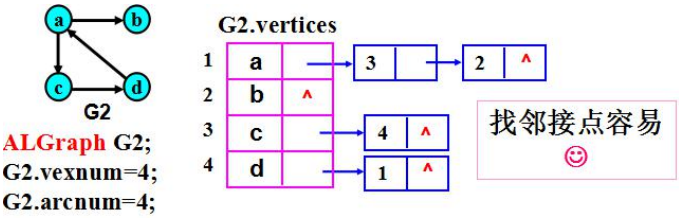
定义：为图中每个顶点建立一个**单链表**，第 i 个单链表中的结点表示依附于顶点 Vi 的边（有向图中值以 Vi 为尾的弧）

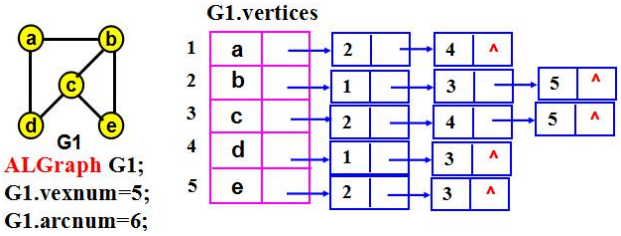
特点：无向图中，单链表的所有结点个数=边数*2；有向图中，单链表的所有结点个数=弧数

无向图中顶点 Vi 的度是第 i 个单链表中的结点数；

有向图中顶点 Vi 的出度是第 i 个单链表中的结点数，入度是整个单链表中邻接点域为 i 的结点个数

具体实例如下：



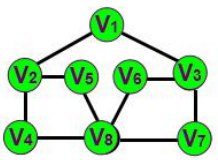


（4）图的遍历

①深度优先遍历（DFS）---类似于树的先根遍历

❖方法：从图的某一顶点Vo出发，访问该顶点；然后依次从Vo的未被访问的邻接点出发，深度优先遍历图，直至图中所有和Vo相通的顶点都被访问到；

若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止。



图的遍历序列不唯一

深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

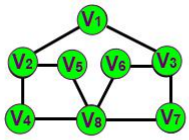
深度遍历：V1⇒ V2 ⇒V5 ⇒ V8 ⇒V7 ⇒V3 ⇒V6 ⇒V4

②广度优先遍历（BFS）---类似于树的层次遍历

❖方法：从图的某一顶点Vo出发，访问该顶点后，依次访问Vo的所有未曾访问过的邻接点；

然后分别从这些邻接点出发，广度优先遍历图，直至图中所有已被访问的顶点的邻接点都被访问到；

若此时图中尚有顶点未被访问，则另选图中一个未被访问的顶点作起点，重复上述过程，直至图中所有顶点都被访问为止



广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

（5）生成树

定义：所有顶点均由边**连通**但**不存在回路**的图，包含 n 个顶点和 n-1 条边

特点：

◆生成树的顶点个数与图的顶点个数相同

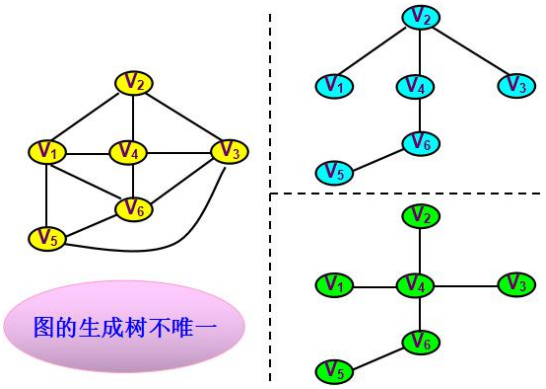
◆生成树是图的极小连通子图

◆一个有n个顶点的连通图的生成树有n-1条边

◆生成树中任意两个顶点间的路径是唯一的

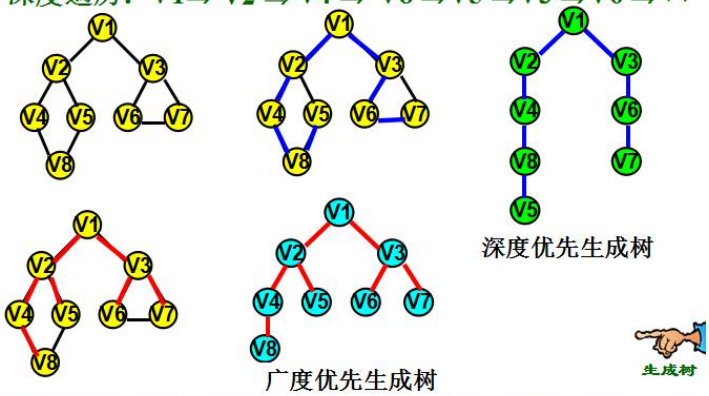
◆在生成树中再加一条边必然形成回路

具体实例如下：



利用深度优先遍历和广度优先**遍历经过的分支**可以**构成生成树**：

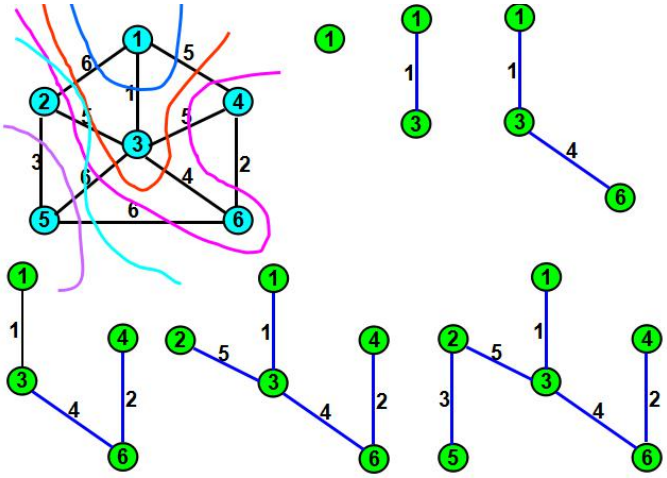
深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7



广度遍历：V1⇒ V2 ⇒V3 ⇒ V4 ⇒V5 ⇒V6 ⇒V7 ⇒V8

构造最小生成树的算法有：Prim 算法、Kruskal 算法

Prim 算法演示：



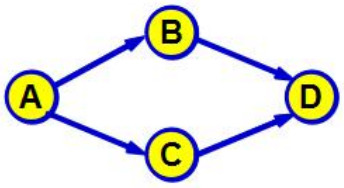
（6）拓扑排序---解决具有优先关系的活动排序问题

★ 定义

- AOV网——用顶点表示活动，用弧表示活动间优先关系的有向图称为顶点表示活动的网(Activity On Vertex network)，简称AOV网
  - 若<v_i,v_j>是图中有向边，则v_i是v_j的直接前驱；v_j是v_i的直接后继
  - AOV网中不允许有回路，这意味着某项活动以自己为先决条件
- 拓扑排序——把AOV网络中各顶点按照它们相互之间的优先关系排列成一个线性序列的过程叫~
  - 检测AOV网中是否存在环方法：对有向图构造其顶点的拓扑有序序列，若网中所有顶点都在它的拓扑有序序列中，则该AOV网必定不存在环

★拓扑排序的方法

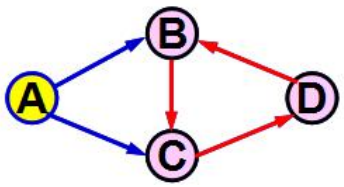
- 在有向图中选一个没有前驱的顶点且输出之
- 从图中删除该顶点和所有以它为尾的弧
- 重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止



拓扑序列：A B C D

或：A C B D

AOV网的拓扑序列不唯一



拓扑序列：A

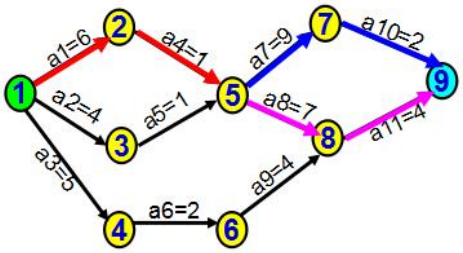
无法[illegible]序列

因为[illegible]

拓扑排序能够检测图中是否有环存在

（7）关键路径

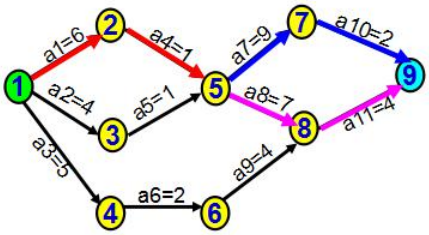
完成整个工程的最短时间为：从有向图的源点到汇点的最长路径。



“关键活动”指的是：该弧上的权值增加，将使有向图上的最长路径的长度增加。

（8）最短路径

完成整个工程的最短时间为：从有向图的源点到汇点的最长路径。



“关键活动”指的是：该弧上的权值增加，将使有向图上的最长路径的长度增加。