1.基本概念

**线性结构**：除第一个和最后一个元素外，任何一个元素都有唯一的**一个直接前驱和直接后继**

**非线性结构：**

树结构：除根，每个结点有且仅有一个直接前驱，有0~多个直接后继，有明显层次关系

图结构：任意元素之间都可能存在关系

**图G=(V,E)** V：顶点集，**非空**有穷集 **故不存在空图**

E：边集，有穷集

设有图G=(V , E)和G’=(V’, E’)

**子图：**V’⊆ V且E’ ⊆ E

**生成子图：**V’= V且E’ ⊆ E

**图的分类**

**按点边关系分类：**

**简单图(simple graph)**：无自环、无重边

**完全图(complete graph)**：**每对顶点**间**都有边**的简单图

**多重图(multigraph)**：有**多重边**连接到**同一对顶点**的图

**按边性质分类：**

**有向图（弧：弧尾-->弧头）、无向图**

**按边/弧的数量将图分类：**

**稀疏图()、稠密图**

**（无向）完全图(有n(n+1)/2条边)、（有向）完全图(有n(n+1)条边)**

**带权图=网**

**路径**

对**无向图**G=(V, E)，若从顶点vi经过若干条边能到达vj，称顶点vi和vj是连通的，又称顶点vi到vj有**路径**

**路径的两种表示：**边/弧序列、顶点序列

**简单路径**：若没有重复顶点的路径

**回路(circuit)/环(cycle)：**第一个顶点和最后一个顶点相同的路径

**简单回路(简单环)：**若除第一个与最后一个顶点之外，其余顶点不重复出现的回路

**连通图**

**对于无向图**

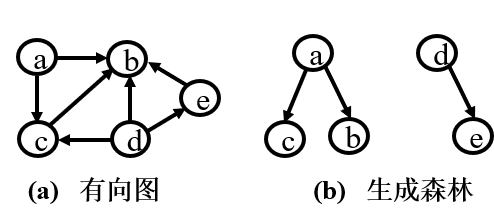
**连通图：**任何两个顶点都是连通的 **非连通图**

**连通分量：**非连通的无向图中的极大连通子图

**极小连通子图：无向连通图的生成树**

生成：覆盖图上所有顶点

树：连通各个顶点但无环。加边的话，出现单环，删边的话，各个顶点不连通

**对于有向图：**

**强连通图：**任何两个顶点都有有向路径连通

**强连通分量：**非强连通的有向图中的极大连通子图

**有向图的生成森林：**由若干棵有向树组成，含有图中**全部顶点**，

但只有足以构成若干棵**不相交**有向树的弧

有向树：只有一个顶点的入度为0 ，其余顶点的入度均为1的有向图

**重连通图**

**重(双)连通图：**若从一个连通图中**删去任何一个顶点**及其相关联的边，它**仍为一个连通图**的话，则该连通图被称为**重(双)连通图**

**关节点/割点：**若连通图中某个顶点和其相关联的边被删去之后，该连通图被**分割成两个或两个以上的连通分量**

**度**

**对于无向图：**

**结点度数TD(vi):** ∀vi∈V，依附于vi的边的数目称为顶点vi的度

**握手定理：所有顶点的度的和是图中边的2倍**，即：∑TD(vi)=2e，e为图的边数

**定理：**在任何图中，所有度数之和必为偶数，度数为奇数的顶点必定是偶数个

**推论：**若图Ｇ有n个顶点，≥n+1条边，则Ｇ中至少有一个顶点的度数大于等于3

**对于有向图：**

**出度**OD(vi)：**以vi作为起点**的有向边(弧)的数目称为顶点vi的**出度(outdegree)**

**入度**ID(vi)：**以vi作为终点**的有向边(弧)的数目称为顶点vi的**入度(indegree)**

顶点vi的出度与入度之和称为**vi的度**，记为TD(vi)，即，TD(vi)=OD(vi)+ID(vi)

**握手定理：**所有顶点的度的和是图中边的2倍

**定理：**在任何有向图中，**所有顶点的入度之和等于所有顶点的出度之和** **∑OD(vi)= ∑ID(vi)**

**2.图的存储结构**

**2.1数组（邻接矩阵）**

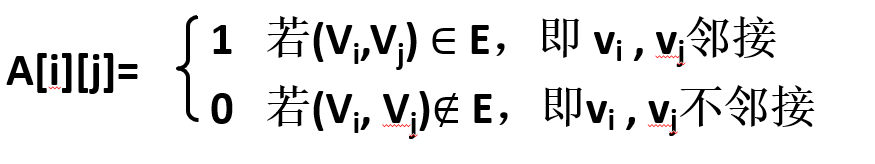
**存储结构MGraph：**对于有n个顶点的图：

顶点数vexnum、边数arcnum

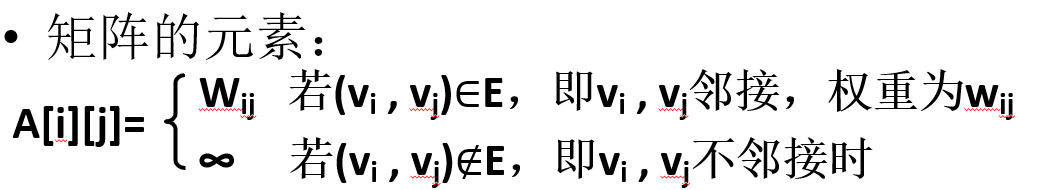
图的种类：UDG无向图，DG有向图，UDN无向网，DN有向网

* + 用**一维数组**vexs[n]存储顶点信息
  + 用**二维数组**A[n][n]存储顶点之间关系信息

无向无权图：



无向带权图：

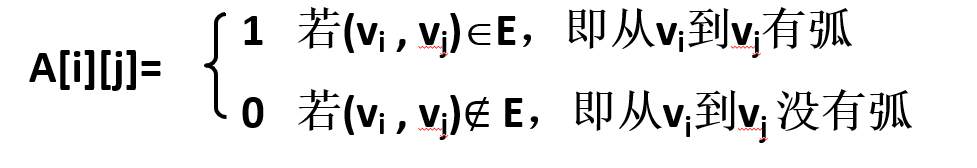


**性质：** 均为**对称矩阵**

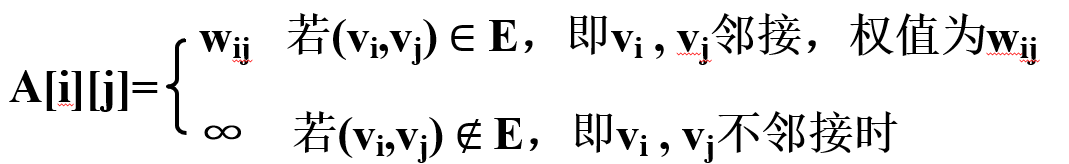
**顶点vi的度数=第i行(或第i列)的非0元素的个数**

**无向图的边数=上(或下)三角形矩阵中非0元素个数**

有向无权图：



有向带权图：



**性质：** 不一定是对称矩阵

**vi出度OD(vi)=第i行的非0元素的个数**

**vi入度 ID(vi)=第i列的非0元素的个数**

**弧的数目=邻接矩阵中非0元素的个数**

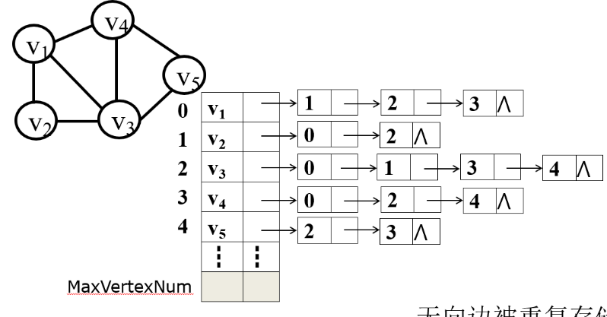
**2.2邻接表法（无向图、有向图的单方向）**

**数据结构AGraph：**

**vexnum顶点数、edgenum边数、kind种类标志**

**表头结点数组v[MAX\_VERTEX\_NUM]**

**为每个顶点建立一个单链表：第i个单链表结点表示依附于顶点Vi的边/弧**

 **adjvex：**与**vi邻接的顶点在表头数组中的位置**

**info**：vi连接到该节点的边/弧的相关信息：权值

**next**：指向下一个表结点

**为每个单链表设一个表头结点：**

**data：**顶点名vi / 或该顶点数据 等能唯一标识该顶点的信息

**first：**指向链表的第一个结点

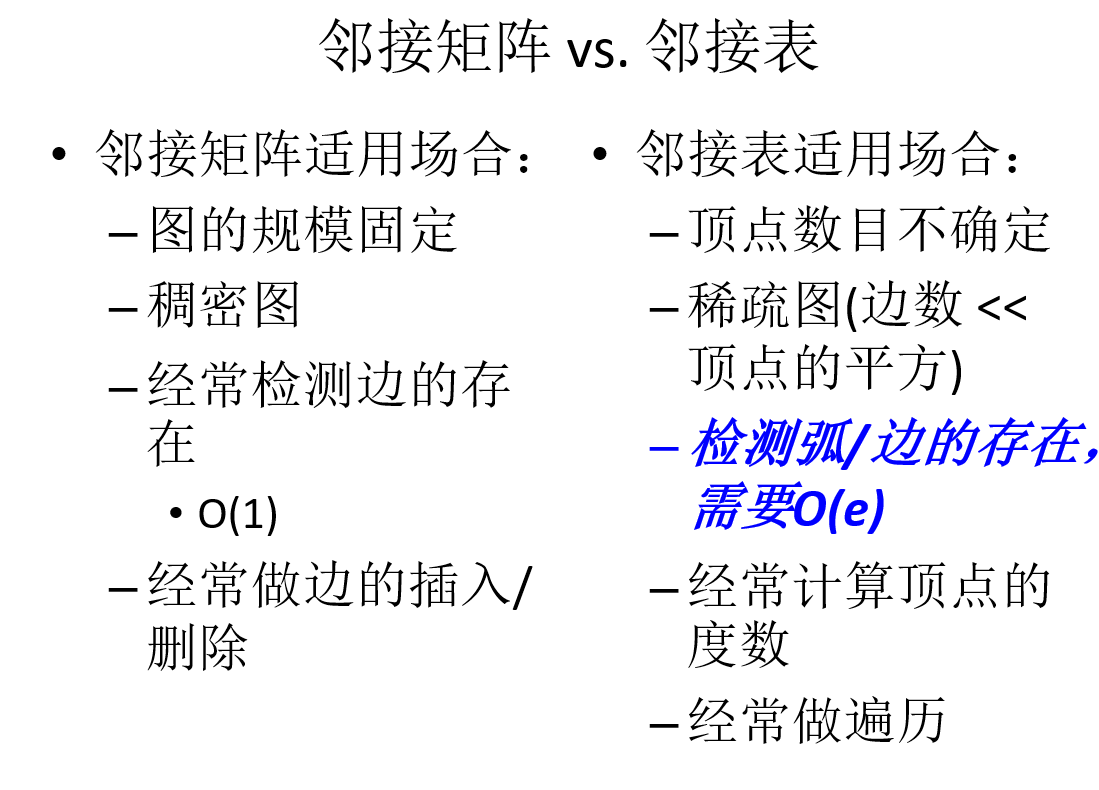
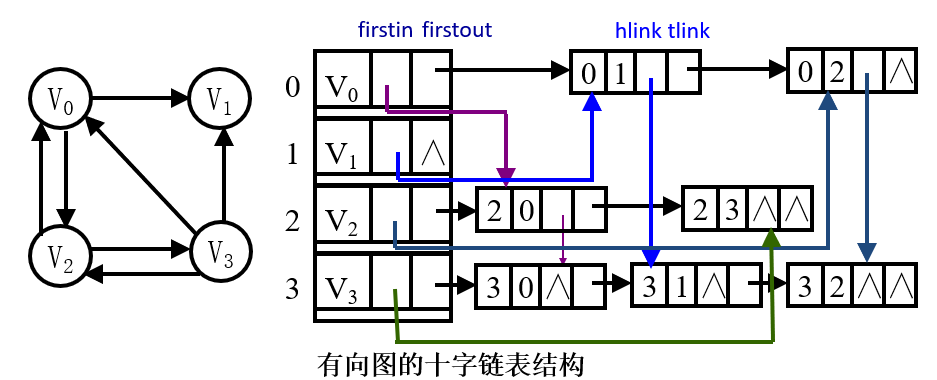
**缺点：无向边会被重复储存**

**特点： 无向图：vi的度=第i个链表的结点数**

**有向图：可构建正邻接表或逆邻接表 正：vi的出度=第i个链表的结点数**

**逆：vi的入度=第i个链表的结点数**

**对于稀疏图，邻接表比邻接矩阵节省存储空间**



**2.3十字链表法（有向图）**

**数据结构OLGraph：**

vexnum,arcnum

xlist[MAX\_VERTEX\_NUM]：VexNode数组

**VexNode：** data

ArcNode\* firstin、\*firstout：指向该顶点的第一条入弧和出弧

**ArcNode：** tailvex、headvex：弧尾和弧头结点在xlist中的位置

(info：弧相关信息：权值)

hlink，tlink：分别连接弧头相同和弧尾相同的弧

**2.4邻接多重表（无向图双向）**

**数据结构AMGraph**：

vexnum，edgenum

adjmulist[MAX\_VERTEX\_NUM]：VexBox数组

**顶点结点VexBox：** data顶点信息

EBox\* firstedge：指向依附于该节点的第一条边

**边结点EBox：** mark访问标记

ivex，jvex：此边依附的两个结点在数组中的位置

ilink：指向依附于顶点i的下一条边

jlink：指向依附于顶点j的下一条边

(info 边相关信息：权值)

所有顶点以一维数组方式组织，储存为顶点结点

每条边用一个边结点表示，用于路径寻找

**3.访问图的顶点：深度优先遍历，广度优先遍历**

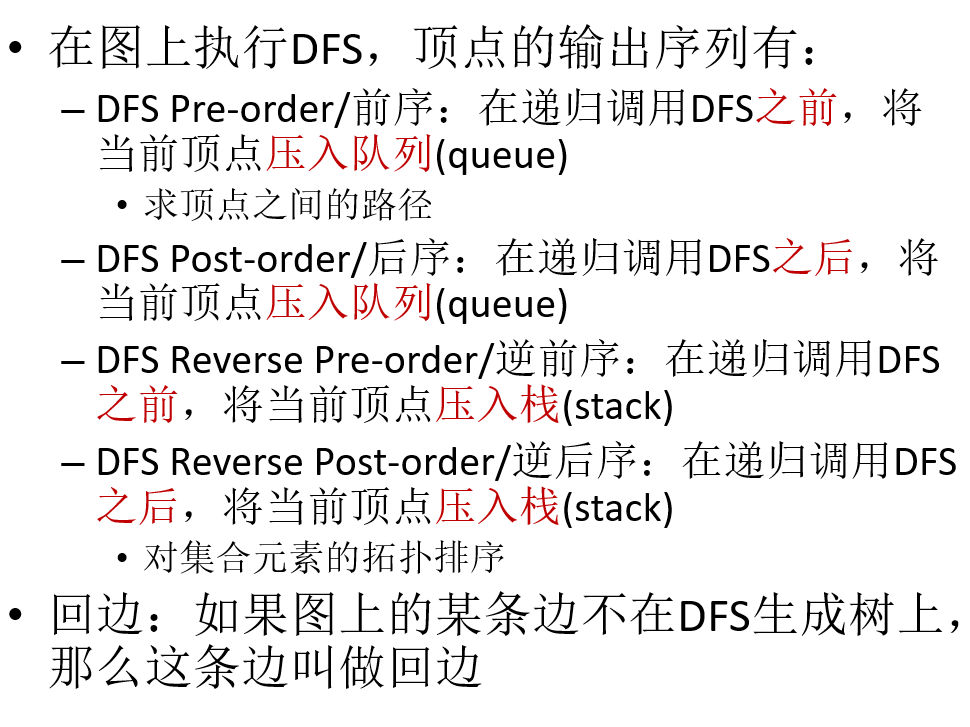
**3.1深度优先搜索**

**访问顶点 V;**

**for (V的邻接点即W1、W2、W3 )**

**若该邻接点W未被访问，**

**则从它出发进行深度优先遍历**

****

**3.2广度优先搜索（基于存储结构）**

从图中的某个顶点V0出发，访问此顶点V0 ，然后依次访问V0的所有未被访问过的邻接点，之后按这些顶点被访问的先后次序（队列）依次访问它们的邻接点，直至图中所有和V0有路径相通的顶点都被访问到

若图中尚有顶点未被访问，则选一个未曾被访问的顶点作为起点，重复上述过程，直到图中所有顶点都被访问到为止

**3.3 图遍历的应用**

**应用1：求从一顶点到另一顶点的一条简单路径**

深度优先搜索，可回溯记录路径

**应用2：求两个顶点之间的一条最短路径**

广度优先搜索

修改队列结构和操作：

将链队列的结点改为“双链”结点：结点中包含next 和prior两个指针

修改出队列的操作：出队列时，仅移动队头指针，而**不将队头结点从链表中删除**

修改入队列的操作：插入新的队尾结点时，**令其prior域的指针指向刚刚（？将要？）出队列的结点**，即当前的队头指针所指结点；

**4.排序(有向) 图的顶点：DAG/AOV网的拓扑排序**

拓扑排序(Topological Sort)：由某个集合上的一个**偏序**得到该集合上的一个**全序**的操作

偏序：自反、反对称、传递。可表示为有向无环图

* AOV网(Activity On Vertex Network)：在有向图中，用**顶点**表示**活动**，用**有向边**表示**活动之间的优先关系**
* 算法思路
  + 选择**一个入度为0(没有前驱)的顶点**并输出
    - 在有多个入度为0的顶点的情况下，选择不同的顶点，会产生不同的拓扑排序结果
  + 删除该顶点以及从该顶点出发的(**以该顶点为尾**的弧)所有有向边
  + 重复执行前两步，直到：
    - 图中全部顶点都已输出(图中无环)，或，
    - 图中不存在无前驱的顶点(图中必有环)

弧尾v🡪 w弧头

* 算法实现
  + 采用**正邻接链表**作为DAG/AOV网的存储结构
  + 设立**栈**，用来暂存入度为0的顶点
    - 也可以用队列来存储
  + 需要记录顶点的入度
    - **删除以该顶点为尾的弧：弧头顶点的入度减1**
  + 需要记录已输出顶点的数目
    - 等于图的顶点数，则，图中无环，否则，图中有环

**5.图的连通性**

**5.1无向图的连通性**

无向非连通图的**连通分量**(**极大**连通子图)

无向非连通图的**生成森林**：由该非连通图内的多个连通图上的生成树(极小连通子图)组成

无向连通图的生成树的构造：

从G中任意顶点出发进行遍历操作，保留所有被经过的边

按DFS算法得到生成树G’称为**深度优先生成树**

按BFS算法得到的G’称为**广度优先生成树**

无向非连通图的生成森林构造：

图存储结构：**邻接表**

生成森林存储结构：**孩子兄弟链表**

* + 从某个**顶点**出发构造图的**深度优先生成树**算法
    - 首先从某个顶点V出发，建立生成树树根结点，然后再分别以V的邻接点为起始点，建立相应的子生成树，并将其作为V 结点的子树链接到V结点上
  + 直到所有顶点都被遍历（可用一个计数器）
    - 算法是一个递归算法

**5.2 有向图的强连通分量**

有向图的强连通分量：

* + 设从V可到达 (以V为起点的所有有向路径的终点)的顶点集合为T1(G)，而到达V (以V为终点的所有有向路径的起点)的顶点集合为T2(G)，则**包含V的强连通分量的顶点集合**是： T1(G)∩T2(G)

有向连通图的深度优先生成树森林的构造：

**​​1. 初始化 ​​：**为每个顶点创建发现标记数组（如visited[]），初始化为未访问（false）。创建一个待访问栈，用来存放待访问的结点。任意选择起始顶点​​入栈，开始以下操作：

**​​2. 递归遍历与树构建​​：** 对于栈顶结点，标记其visited值，检查其出边邻接点访问状态：

若找到第一个未标记的邻接点，则将该结点入栈，重复操作；

若都已被标记，则将栈顶结点出栈并访问此结点（入访问栈，用于记录访问顺序），重复操作直到栈空；

若栈空，任意选择一个还未标记过的顶点，重复以上构造 **（访问顺序≠发现顺序）**

**​​3. 生成树结构​​**

**​​**树节点关系​​：每个节点在生成树中仅有一个父节点（起始节点无父节点），其**子节点按DFS访问顺序排列，形成类似左孩子右兄弟的树结构。**

**​​多连通分量处理​​：若图存在多个连通分量，需对每个未访问的顶点重复上述过程，形成多棵生成树（森林）**

**Kosaraju算法**：求有向图G的强连通分量

一个图和它的逆图(边全部反向形成的图，transpose graph, reverse graph)具有相同的强连通分量

（1）深度优先进行第一次遍历，按**访问顺序**逆后序排列：即再维护一个栈，每访问一个结点便入栈

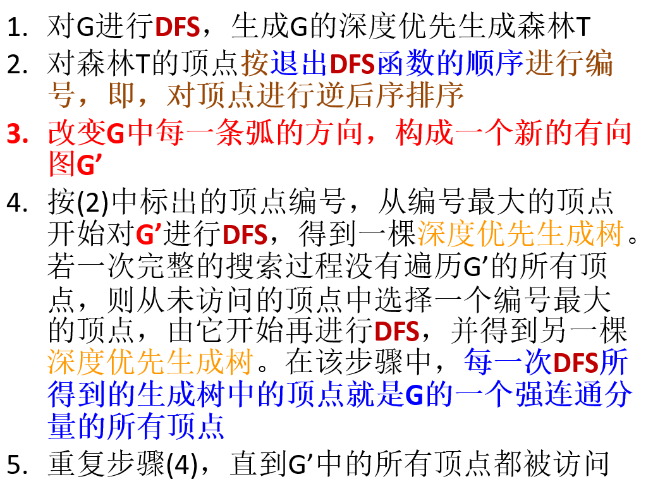
（2）重置访问标记

（3）每次取栈顶结点，检查其是否访问过：

若没有，访问该节点，并以其为起点进行深度优先遍历；

若有，跳过该节点重复此步，直到栈空为止。

每次深度优先遍历的一组结点即为一个强连通分量



**5.3 重连通图和关节点**

重连通图：若从一个连通图中删去任何一个顶点及其相关联的边，它仍为一个连通图

关节点分析：需借助图的深度优先生成树来分析

* 若生成树的**根结点**有两个或两个以上的**分支**，则此顶点(**生成树的根**)必为**关节点**
* 对生成树上的任意一个**内部结点v(非叶子结点)**，若其某棵子树的根或子树中的其它结点**没有和v祖先相通的图上的回边，**则该**结点v**必为**关节点**
  + **顶点v为关节点：**
    - 存在顶点**w** 是生成树上顶点**v** 的**孩子，且**
    - **w及其子孙均无指向v的祖先的回边**

**5.4 最小生成树MST**

覆盖G中的所有顶点，连通各个顶点但无环

最小代价生成树：生成树各边权重之和最小

**Prim算法**

* 从连通网N=(V, E)中找最小生成树T = (U, TE)
  + 若从顶点v0出发构造，U = {v0}，TE={}；
  + **先找权值最小的边(u, v)，其中u∈U且v∈V-U**，然后U = U∪{v}，TE=TE∪{(u, v)} ；
  + 重复前一步，直到U=V为止，这时，TE中必有n-1条边，T=(U, TE)就是最小生成树

**实现：**

**数据结构MGraph邻接矩阵**

closedge[ ]：用来保存V- U中各顶点到U中顶点具有权值最小的边

struct {

Elemtype adjvex; //边所依附于U中的顶点

int lowcost ; //该边的权值

} closedge[MAX\_VERTEX\_NUM];

* 从closedge中选择一条权值(不为0)最小的边(vk, vj) ，加入树中/打印，然后：
  + 置closedge[k].lowcost为0，表示vk已加入到U中
  + 根据新加入的顶点vk，更新closedge中每个元素：
    - ∀vi∈V-U ，若cost(i, k) < colsedge[i].lowcost，表明在U中新加入顶点vk后， (vi, vk)成为vi到U中权值最小的边，那么做如下设置：
    - closedge[i].lowcost=cost(i, k)
    - closedge[i].adjvex=k

**Kruskal算法**

在**拟阵**上实施贪心策略，能实现最优解

* 设G=(V, E)是具有n个顶点的连通网，现要构造G的最小生成树 T=(U, TE)
  + 初始时：U=V，TE={}
  + 对G中的边**按权值大小**从小到大排序
  + **选取权值最小的边**(vi,vj)
  + 若边 (vi,vj) 加入到TE后形成**回路**，则舍弃该边；否则，将该边并入到TE中，即TE=TE∪{(vi, vj)}
  + 重复前一步骤，**直到TE中包含有n-1条边为止**

**贪心算法(Greedy algorithm)**

在对问题求解时，总是做出**在当前看来是最好的选择**

往往得到局部最优解

**图的拟阵**

**拟阵：**

* + S是一个有穷非空集合
  + (的遗传性)是S的一个非空子集族，称为S的独立子集，使得如果且，那么
  + (M的交换性) 如果， ，且|A|<|B|，则有某个元素使得

**定理**：如果G=(V,E)是一个无向图，则是个拟阵，其中集合为E，即G的边集，为无回路的E的子集

* **贪心算法能求得最优独立集的充分必要条件是该独立系统是一个拟阵**

**6.(带权)图的路径**

**6.1 AOE网的关键路径**

AOE网**(Activity On Edge Network)**是一个**有向无环带权图**

* + 只有一个入度为0的点(**起点**)和一个出度为0的点(**汇点/终点**)
  + **顶点：事件(Event)**、**弧：活动(Activity)**
  + 弧上的**权值：活动所需的时间(Duration)或费用**

**关键路径(Critical Path)：**从起点到汇点的长度最长的路径。其长度表示完成整个工程所需的最短时间。其上的活动为关键活动，关键活动若不按期完成，就会影响整个工期。

非关键活动加速并不能使整个工程提前。

**事件(顶点)(经过关键路径的)最早发生时间ve：是从起点到该顶点的最长路径长度**

ve(j) = Maxi { ve(i)+dut(<i, j>) } i为j的所有前驱

汇点的最早发生时间是最短工期（不影响进度的情况下的工期）

**事件(顶点)(不影响整体工程进度的)最迟发生时间vl：**汇点的最早发生时间 **减** 该顶点到汇点的**最长**路径长度

vl(j) = Min{vl(k)-dut(<j, k>) } k为j的所有后继

**起点、汇点**的**最早发生时间等于其最迟发生时间**（必须从起点开工、必须在汇点完工）

**vl(s)=ve(s)=0 vl(t)=ve(t)**

**理解：**

最早发生时间：沿关键路径一路走到该结点，前面的事件都不能有任何拖延才能在ve时刻开始该事件，即ve是最早发生时间

最迟发生时间：为了不影响工期，需要在终点事件最早发生时间之前到达终点，即为了让关键路径上没有任何工程拖延，该事件最晚在vl时刻开始。

**弧(活动)的活动最长可持续时间：**其**弧头**的**最迟发生时间**减去**弧尾**的**最早发生时间**

**弧(活动)**允许的**延时D：**弧的活动最长可持续时间减去**弧权**

1->2 D=vl(2)-ve(1)-w

**延时为0的活动为关键活动（关键活动连成关键路径）**

设活动ai对应弧<j, k>

**弧(活动)的最早开始时间e(i) = ve(j)**

**弧(活动)的最迟开始时间 l(i) = vl(k)-dut(<j,k>)**

D= **l(i)-e(i)表示活动ai的时间余量/允许的延时，若l(i)-e(i)=0表示活动ai是关键活动**

**计算：事件(顶点)vj的最早发生时间ev(vj)**

1. 起点事件的最早发生时间设为0

2. 除起点外，对于顶点vj，只有vj的所有前驱事件vi的最早发生时间ve(i)计算出来后，才能计算ve(j)

方法：对所有事件(顶点)进行拓扑排序，然后按拓扑顺序，依次计算每个事件的最早发生时间

**计算：事件(顶点)vj的最晚发生时间el(vj)**

只有vj的所有后继事件vk的最迟发生时间vl(k)计算出来后，才能计算vl(j)

方法：按拓扑排序的逆序，依次计算每个事件的最晚发生时间

**算法思想-识别关键路径CriticalPath**

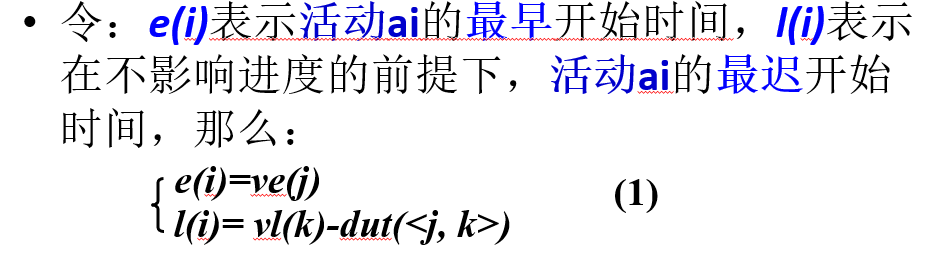
1. 利用拓扑排序计算图g的一个拓扑序列

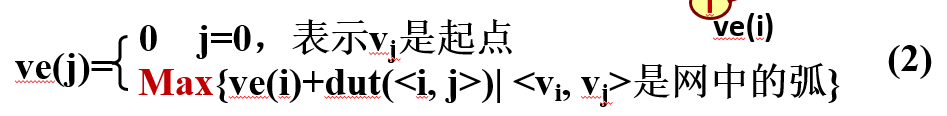
2. 从拓扑排序序列的第一个顶点(起点)开始，按顺序计算ve(i)

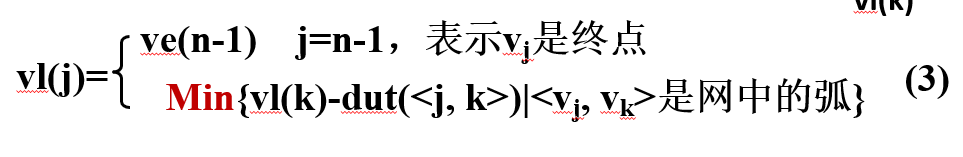
实际上：Status TopologicalOrder(AGraph \*g, Stack &T, int ve[])同时获取ve和拓扑排序序列。有向网g采用邻接表存储结构，若g无回路，则用栈T(即：char orderedV[MAX\_VERTEX\_NUM])返回g的一个拓扑序列，ve中保存各顶点事件的最早发生时间

3. 按拓扑顺序的逆序，从最后一个顶点(汇点)开始，依次计算vl(i)

4. 计算e(i),l(i)，找到l(i)-e(i)=0的活动，并输出

弧i的起点为j，终点为k





**整个算法的时间复杂度是O(n+e) 顶点数+边数**

**6.2 最短路径**

**单源点的最短路径(Single source shortest path)**

最短路径的特征-定理：针对同一个源点的其余最短路径，或者是直接从源点到该点(只含一条弧)；或者是，从源点经过已求得最短路径的顶点，再到达该顶点

**Dijkstra算法**：一种**按路径长度递增次序**产生最短路径的算法（可用于有向或无向图）

数据结构： 给定原点v0

* **n顶点有向图: 用邻接矩阵g[n][n]表示**
* **S是已求得最短路径的终点的集合**
* 数组**final[0..n-1]**用于标识一个顶点是否已加入S中
* 数组**dist[0..n-1]**保存了当前已求得的从源点（只经过集合S中的顶点）到其余各顶点的最短路径值
  + dist[i]的初始状态：g[v0][vi] （有边为权值，无边为∞）
  + **dist[i]更新**：=Min{dist[i], dist[k](Vk∈S)和弧(Vk,Vi)上的权值之和，Vi∈V-S}

**算法描述**：

0. 用**邻接矩阵**g[n][n]表示有向图

n为图中顶点个数

g[i][j]表示弧(i,j)上的权重

1. 初始化：

**S←{v0};**

dist[j]←g[0][j], j = 1, 2, …, n-1;

2. 求出最短路径的长度：**找到V-S中dist最小的结点，将其加入S**

dist[k]←min {dist[i], vi∈V-S} ; k为使dist最小的i

S←S∪{k};

3. 修改： 根据已经被确定为最短路径的S更新**V-S中的dist**

对于每一个 **i∈V-S**;

dist[i]←min{dist[i], dist[k]+g[k][i]}, **注意k∈S**

4. 判断：若 S = V，则算法结束；或者本次没有新节点加入S，有多个连通分量，需递归调用。否则转2.继续执行。

**时间复杂度是O(n2)**

**使用二维数组P保存从v0顶点到其余顶点v的最短路径**

P第i行上的非零元素：表示从v0到vi的最短路径上包含这些列号所对应的结点

**Dijkstra算法：**权值非负情形的单源最短路径(Single source shortest path)问题（可以有回路）

下面需有(若图上无包含由带负权值的边组成的回路)

**Bellman-Ford算法：**边上权值为任意值的单源最短路径问题

**Floyd算法：**边上权值为任意值的所有顶点之间的最短路径

**Floyd算法O(n3)**

