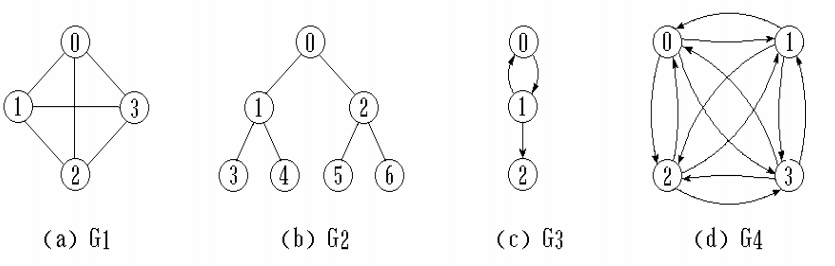
一.图的定义

图是由定点集合(Vertex)以及顶点间的关系集合组成的一种数据结构

Graph = (V,E)

V={X|X}是顶点的有穷非空集合

E= 是顶点集合的有穷集合，也叫边集合



二.相关概念

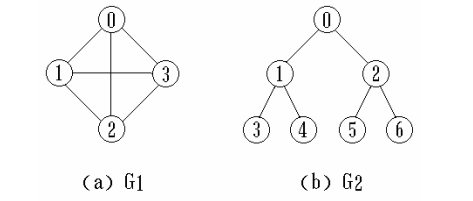
1.无向边

若顶点x和y之间的边没有方向，则称该边为无向边（x,y）

(x,y)与（y,x）意义相同，表示x和y之间有连接

2.无向图

若图中任意两个顶点之间的边均是无向边，则称该图为无向图



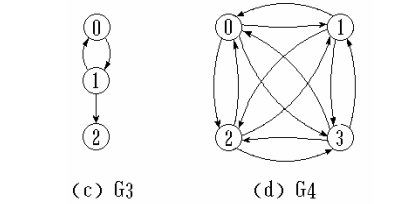
3.有向边

若顶点x和y之间的边有方向，则称该边为有向边<x,y>

<x,y>与<y,x>意义不同，<x,y>表示从x连接到y,x为尾，y为头

有向图

如图中任意两个顶点之间的边均是有向边，则称该图为有向图



4 度的定义

顶点V的度是和v相关联的边的数目，记为TD(v)

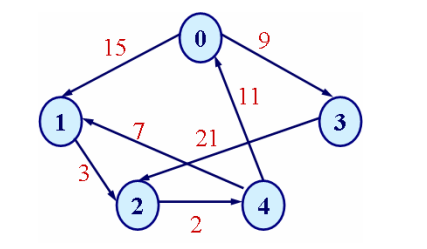
入度： 以v为头的边的数目，记为ID(V)

出度： 以v为尾的边的数目，记为OD(v)

5 权的定义

与图的边相关的数字叫做权

权常用来表示图中顶点间的距离或者耗费



图的一些常用操作

创建图

销毁图

清空图

加入边

删除边

获取权

获取结点的度

获取图的结点数

获取图的边数

三.图的存储结构

1 邻接矩阵法：

用一维数组存储顶点（描述顶点相关的数据）

用二维数组存储边（描述顶点间的边）

设图 A = (V,E)是一个有n个顶点的图，图的邻接矩阵为Edge[n][n]

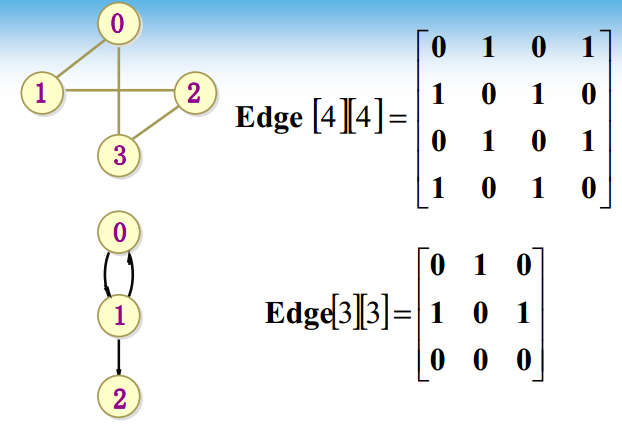
则：

w,w>0 i和j连接

Edge[i][j] =

0 i==j或i和j不连接

w是权值，当不需要权值时，取W为1表示结点间的连接



邻接矩阵头结点

记录顶点个数

记录与顶点相关的数据描述

记录描述边集的二维数组

typedef struct \_tag\_MGraph{

int count;

MVertex\*\* v;

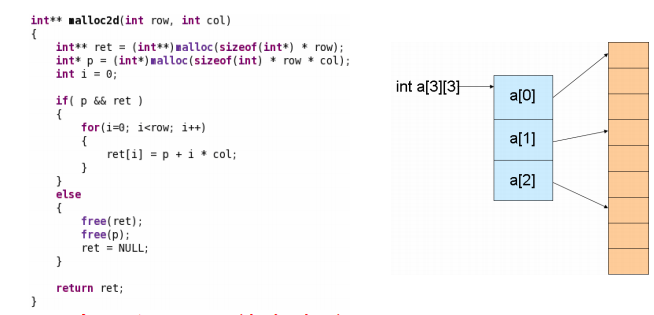
}TMGraph

动态申请二维数组

通过二级指针动态申请一维指针数组

通过一级指针申请数据空间

将一维指针数组中的指针连接到数据空间

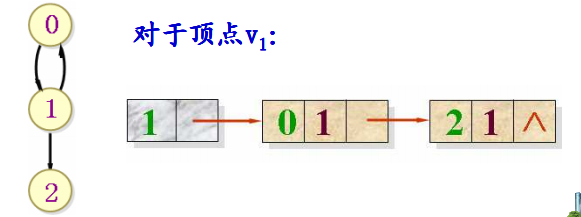


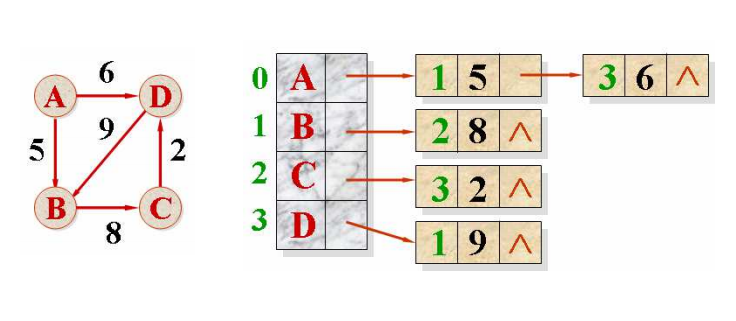
2.邻接链表法

1.基本思想

从一个顶点发出的边连接在同一个链表中

每一个链表结点代表一条边，结点中保存边的另外一个顶点的下标和权值





邻接链表的头结点

1.记录顶点个数

2.记录和顶点相关的数据描述

3.记录描述边集的链表数组

typedef struct \_tag\_LGraph

{

int count;

LVertex\*\* v;

LinkList\*\* la;

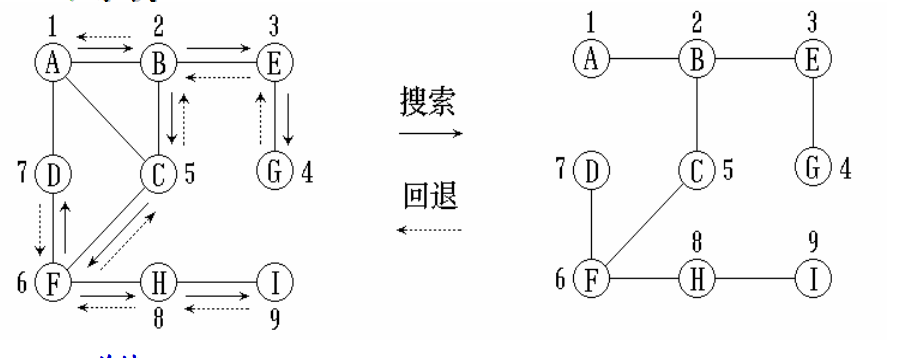
}

四.图的遍历

1定义

从图中的某一个顶点出发，沿着一些边访遍图中所有的顶点，使得每个顶点仅被访问一次

2 深度优先遍历（DFS）



算法：1.访问起始顶点v

1.1当v还有邻接顶点未访问时，深度遍历未访问过的邻接顶点w

1.2当v的所有邻接顶点都被访问时

如图中所有顶点均已被访问，算法结束

如图中还有未访问的顶点，以未访问顶点作为起始顶点深度遍历

过程中需要一个标记顶点是否被访问的辅助数组 visited[]

3 广度优先遍历

算法：1. 访问起始顶点V0

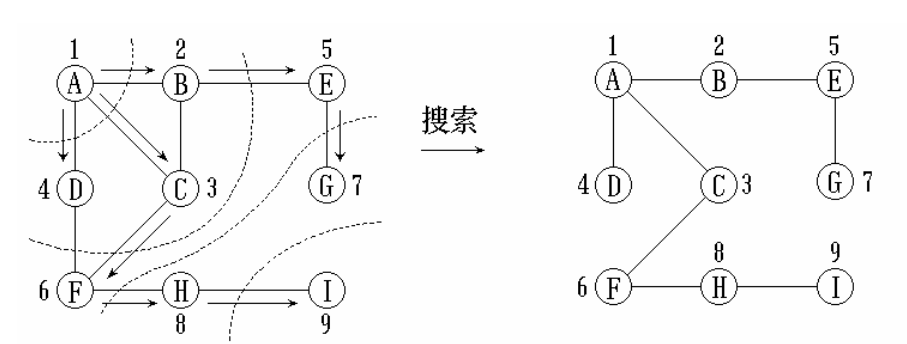
2. 依次访问V0的各个邻接点V01，V02，……V0x

3. 假设最近一次访问的顶点依次为Vi1,Vi2,……,Viy

则依次访问Vi1,Vi2,……,Viy的未访问的结点

4. 重复3，直到所有顶点均被访问

广度优先遍历是一种层次遍历，需要借助队列实现



同样需要一个标记顶点是否被访问过的辅助数组 visited[]

五.最小连通网

1.特点：

使用图中的n-1条边来连接n个顶点

不产生回路

各边上的权值总和达到最小

2.Prim算法

基本思想

从图 N = {V,E}中选择某一顶点进行标记，之后选择与它关联的具有最小权值的边（）,并将顶点v进行标记

反复在一个顶点被标记，而另外一个顶点未被标记的各条边中选择权值最小的（u,v）

并将未标记的顶点进行标记

循环上面的过程，直到图中的顶点都被标记为止

算法步骤

1.从某一项出发，使得U={}，TE={}

2.每一次选择一条边，这条边是所有（u,v）中权值最小的边

且u,修改U和TE：

TE = TE+{（u，v）}

U = U+{v}

3.当U，转2,；否则结束

Prim算法针对顶点展开，适合于边的数量较多的情况

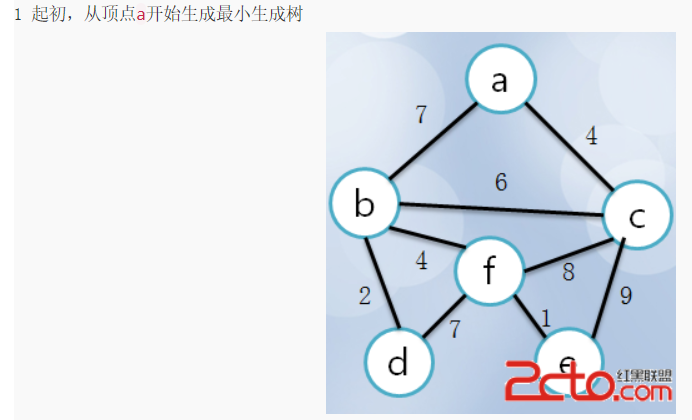
总结：Prim算法从任意一个顶点开始，每次选择一个顶点集最近的一个顶点，并将两顶点间的边

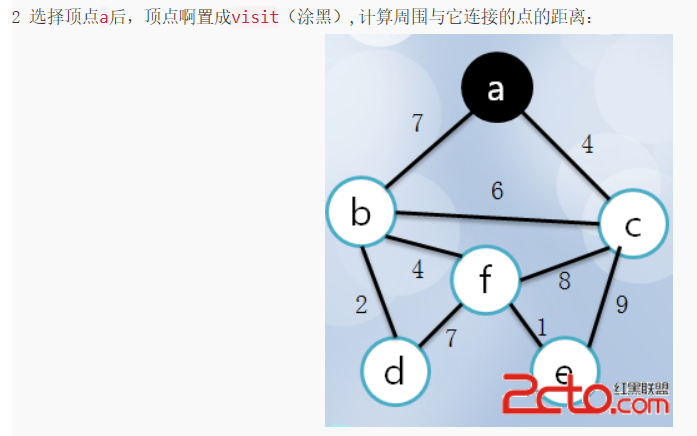
加入到树中（贪心算法）

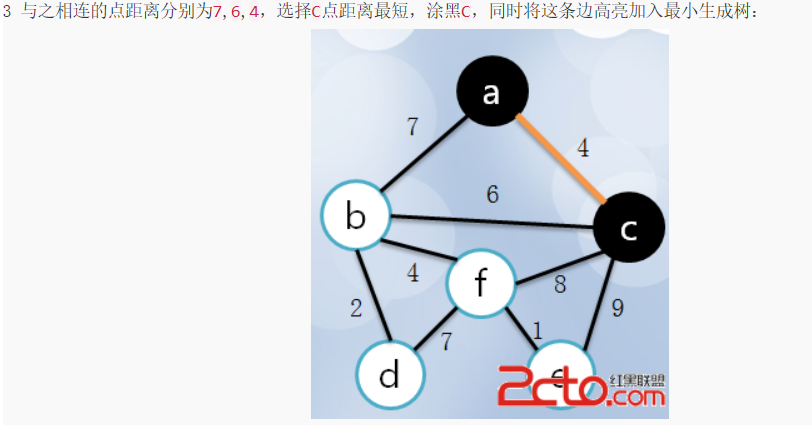
过程： 1. 在一个加权连通图中，顶点集合V，边集合为E

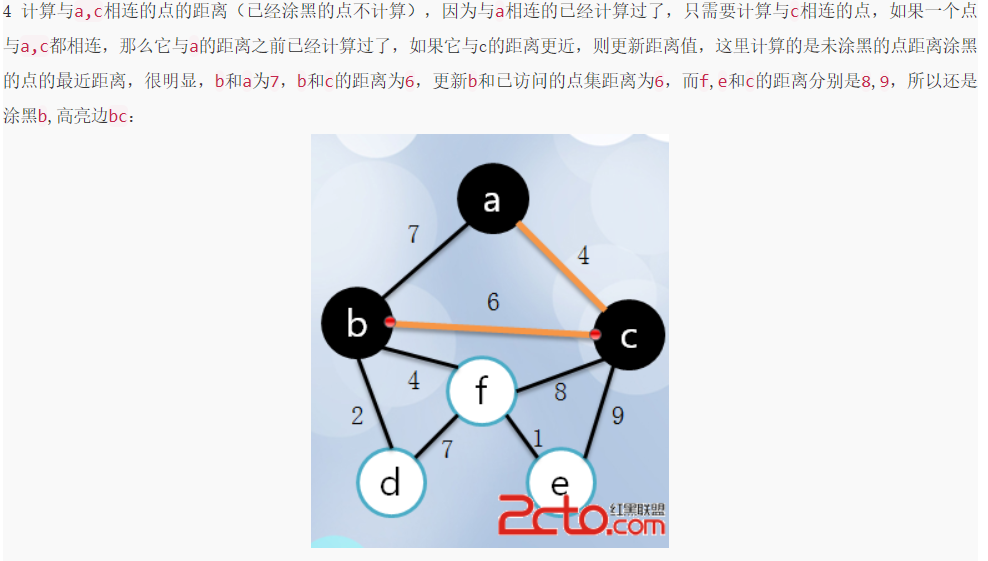
2. 任意选出一个点作为初始顶点,标记为visit,计算所有与之相连接的点的距离，选择距离最短的，标记visit.  
 3. 重复以下操作，直到所有点都被标记为visit：  
 在剩下的点中，计算与已标记visit点距离最小的点，标记visit,证明加入了最小生成树。

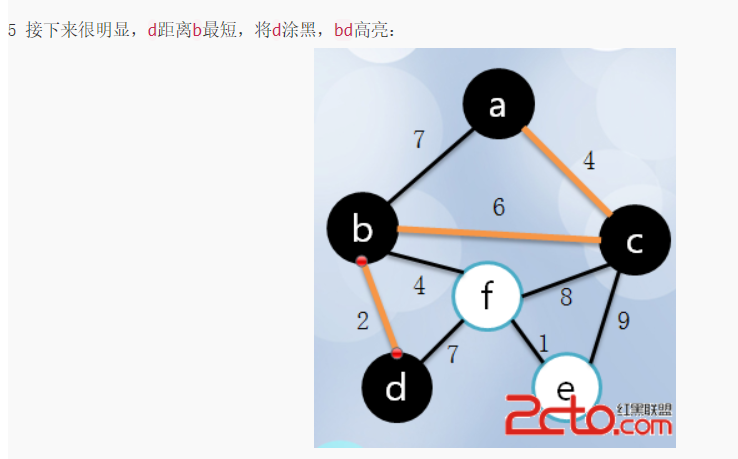
例：

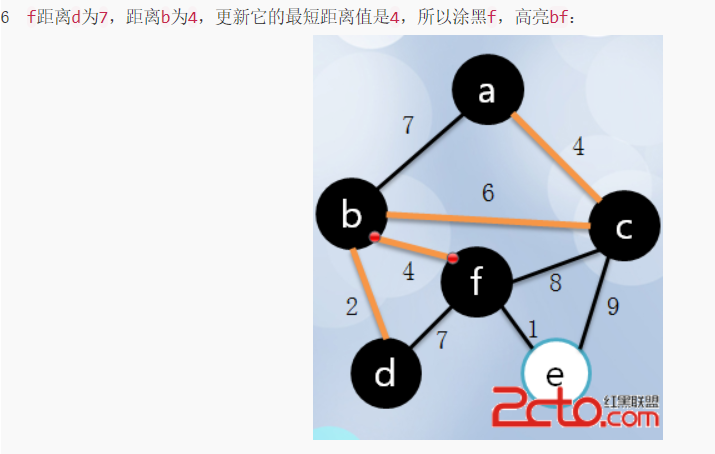


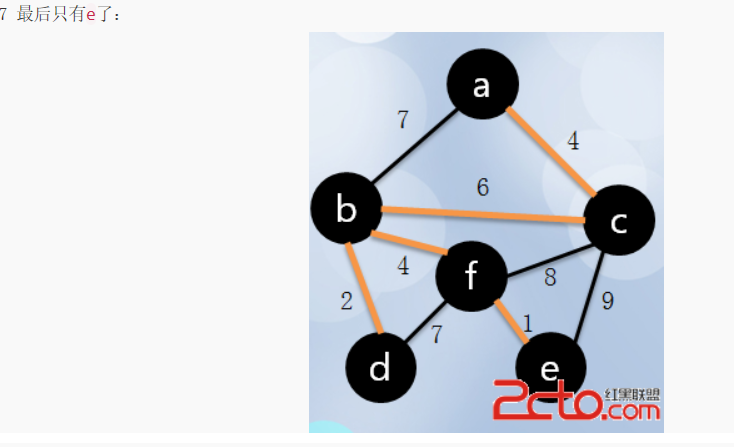












3.Kruskal算法

3.1 基本思想

对于n个顶点的图G={V,E}

构造一个只有n个顶点，没有边的图G'= {V,};

在E中选择一条具有最小权值的边，若该边的两个顶点不构成回路，则将此边

加入到T中，否则将此边舍去，重新选择一条权值最小的边

重复上面的步骤，直到所有的顶点都连通

3.2 实现步骤

1定义边结构体

typedef struct \_tag\_Edge

{

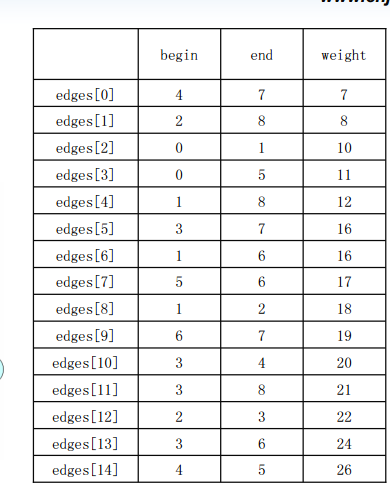
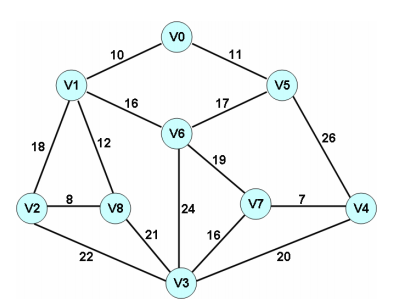
int begin ;

int end;

int weight;

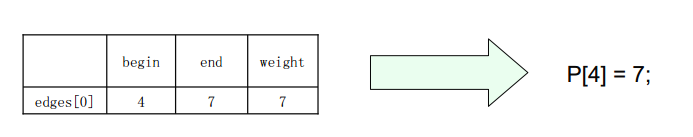
}TEdge

2 定义边集数组并排序

3 定义辅助数组P[n],其中n为顶点数目

P[n]的用于记录边顶点的首尾连接关系



3.3核心思想

遍历edges数组中的每个元素

通过P数组查找begin顶点的最终连接点v1

通过P数组查找end顶点的最终连接点v2

-v1!=v2 ,则

当前边为最小连通网中的边，记录连接关系P[v1]=v2;

-v1==v2 ,则

产生回路，舍弃当前边

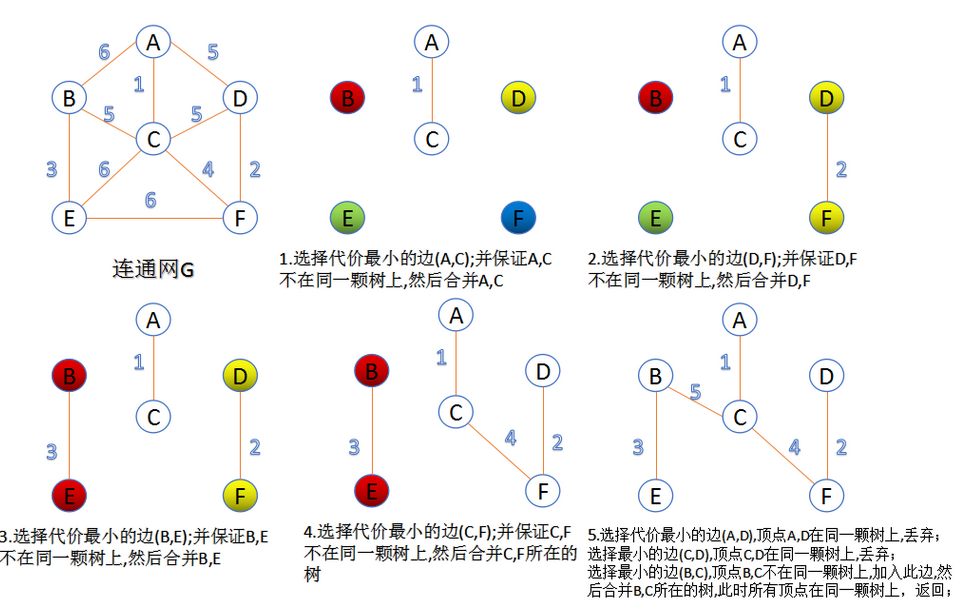
Kruskal算法是针对边展开的，适合于边的数量较少的时候

总结：Kruskal是先将每个顶点放入到各自的数据集合中，然后按照权值的升序来选择边

当选择每条边时，判断定义边的顶点是否在不同的数据集中，如果是将此边插入最

小连通图的集合中，如果不是就移动到另外一条边，重复直到所有边都被探查

例 ：



六.最短路径

1最短路径问题

如果从有向图的某一顶点(称为源点)到达另外一个顶点(称为终点)的路径可能不止一条

如何找到一条路径使得沿此路径上的各边上的权值总和达到最小

即：给定一个带权有向图D与源点v，求从v到D中其它顶点的最短路径。

2.解决方法

2.1 Dijkstra算法

首先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止

2.2 Dijkstra详细步骤

2.2.1 设置辅助数组dist，它的每一个分量dist[i]表示当前找到的从原点

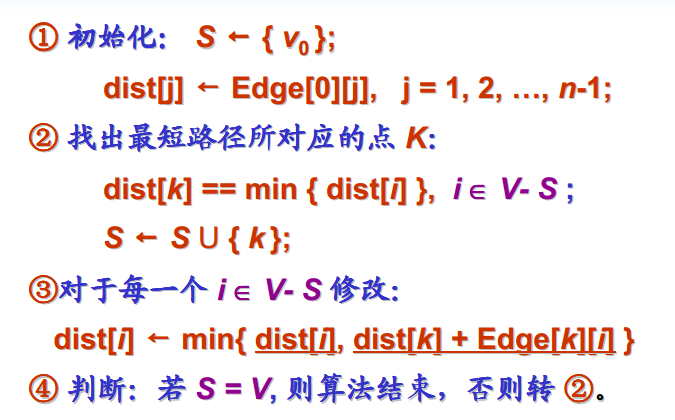
到的最短路径的长度

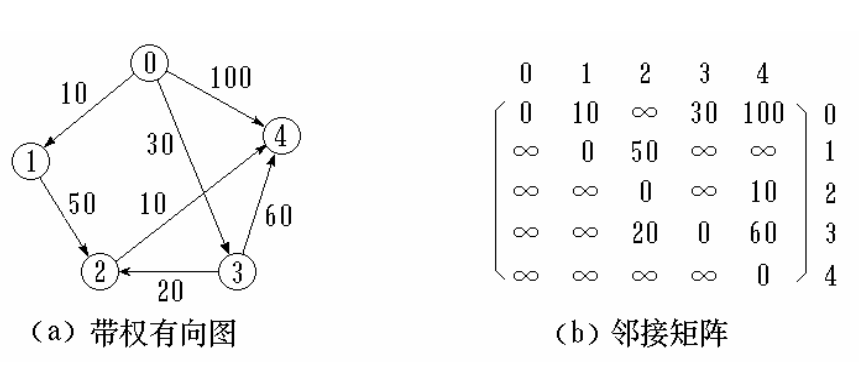
2.2.2 初始状态

若从源点到有边：dist[i]为该边上的权值

若从源点到无边：dist[i]为

2.2.3 步骤：



例：

源点为0

1）初始化 dist = {0,10,,30,100},S = {0}, V={0,1,2,3,4}

2）找出最短点 dist[k] = 10,k=1 ,S ={0,1}

dist = {0,10,60,30,100}

S != V,继续执行

3）找出最短点 dist={0,10,60,30,100}

dist[k] = 30,k=3, S = {0,1,3}

更新dist = {0,10,50,30,90}

S!=V继续执行

4） dist = {0,10,50,30,90} ,S={0,1,3}

最短点 dist[k] = 50,k=2,S = {0,1,3,2}

更新dist = {0,10,50,30,60}

S!=V 继续执行

5）dist = {0,10,50,30,60}，S={0,1,3,2}

只剩下4结点

dist[k] = 60,k=4 S={0,1,3,2，4}

S=V 结束

即0->1->3->2->4的路径最短

总结：下一条长度次短的路径必然是从源点直达或经过上一次已确定结点的路径

Floyd算法

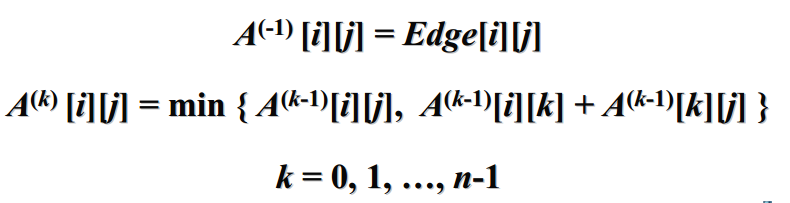
Floyd算法又称为插点法，利用动态规划的思想寻找给定的加权图中多源点之间最短路径的算法

基本思想

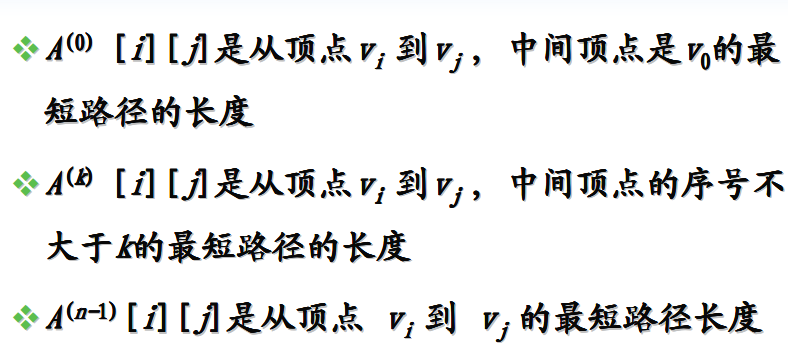
定义一个n阶方阵序列



其中



A矩阵的意义



算法过程

1. 从任意一条单边路径开始，所有两点之间的距离是边的权，如果两点之间没有边相连

则权为无穷大

2. 对于每一对顶点u和v，看看是否存在一个顶点w使得u到w再到v比已知的路径

更短，如果是更新它

例：

