**6.1图的基本概念**

* 图定义：图（Graph）是由**顶点的有穷非空集合**和顶点之间**边的集合**组成，通常表示为：**G(V,E)**，其中，**G**表示一个图，**V**是图G中顶点的集合，**E**是图G中边的集合：

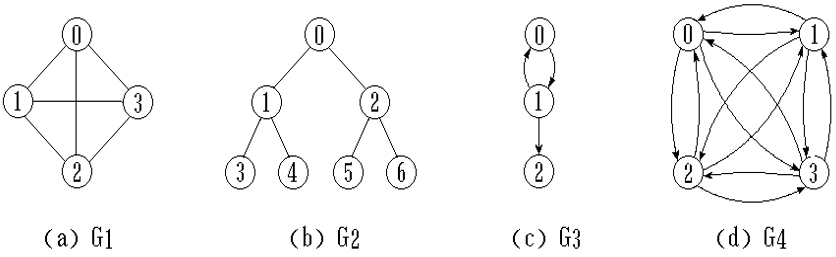
***Graph=（V,E）***

其中 V={x|x某个数据对象} 是顶点的有穷非空集合；

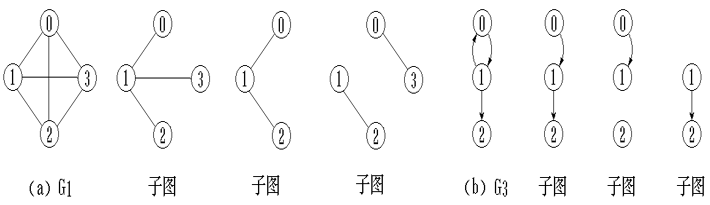
E={(x,y)|x,yV}

或 E={<x,y>|x,yV&&Path(x,y)} 是顶点之间关系的有穷集合，也叫做边(edge)集合。Path(x,y)表示从x到y的一条单向通路,它是有方向的。

* 有向图与无向图：在有向图中，顶点对<x,y>是有序的。在无向图中，顶点对(x,y)是无序的。
* 完全图：若有n个顶点的无向图有n(n-1)/2条边，则此图为完全无向图。有n个顶点的有向图有n(n-1)条边，则此图为完全有向图。

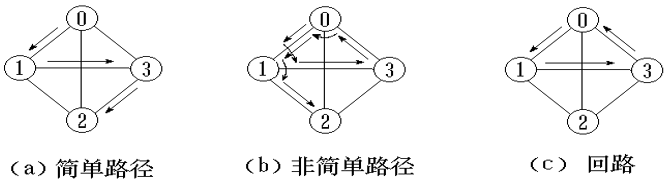


* 邻接顶点：如果(u,v)是E(G)中的一条边，则称u与v互为邻接顶点。
* 权：某些图的边具有与它相关的数,称之为权。这种带权图叫做网络。
* 子图：设有两个图G＝(V,E)和G‘＝(V’,E‘)。若V’V且E‘E,则称图G’是图G的子图。

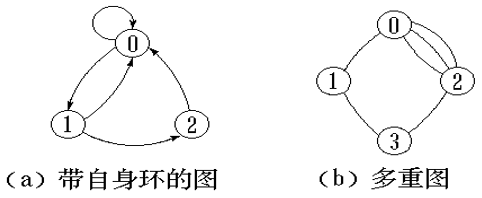


* 顶点的度：一个顶点v的度是与它相关联的边的条数。记作TD(v)。在有向图中,顶点的度等于该顶点的入度与出度之和。
* 顶点v的入度：是以v为终点的有向边的条数,记作ID(v);顶点v的出度是以v为始点的有向边的条数,记作OD(v)。
* 路径：在图G＝(V,E)中,若从顶点vi出发,沿一些边经过一些顶点vp1,vp2,…,vpm，到达顶点vj。则称顶点序列(vivp1vp2...vpm vj)为从顶点vi到顶点vj的路径。它经过的边(vi,vp1)、(vp1,vp2)、...、(vpm,vj)应是属于E的边。
* 路径长度：
* 非带权图的路径长度是指此路径上边的条数。
* 带权图的路径长度是指路径上各边的权之和。
* 简单路径：若路径上各顶点v1,v2,...,vm均不互相重复,则称这样的路径为简单路径。

回路：若路径上第一个顶点v1与最后一个顶点vm重合,则称这样的路径为回路或环。



* 连通图与连通分量：在无向图中,若从顶点v1到顶点v2有路径,则称顶点v1与v2是连通的。如果图中任意一对顶点都是连通的,则称此图是连通图。非连通图的极大连通子图叫做连通分量。
* 强连通图与强连通分量：在有向图中,若对于每一对顶点vi和vj,都存在一条从vi到vj和从vj到vi的路径,则称此图是强连通图。非强连通图的极大强连通子图叫做强连通分量。
* 生成树：一个连通图的生成树是它的极小连通子图，在n个顶点的情形下，有n-1条边。但有向图则可能得到它的由若干有向树组成的生成森林。
* 本章不予讨论的图



**6.1.1图的抽象数据类型**

class Graph{

public:

Graph();

void InsertVertex(const Type&vertex);

void InsertEdge

(const int v1,const int v2,int weight);

void RemoveVertex(const int v);

void RemoveEdge(const int v1,const int v2);

int IsEmpty();

Type GetWeight(const int v1,const int v2);

int GetFirstNeighbor(const int v);

int GetNextNeighbor(const int v1,const int v2);

}

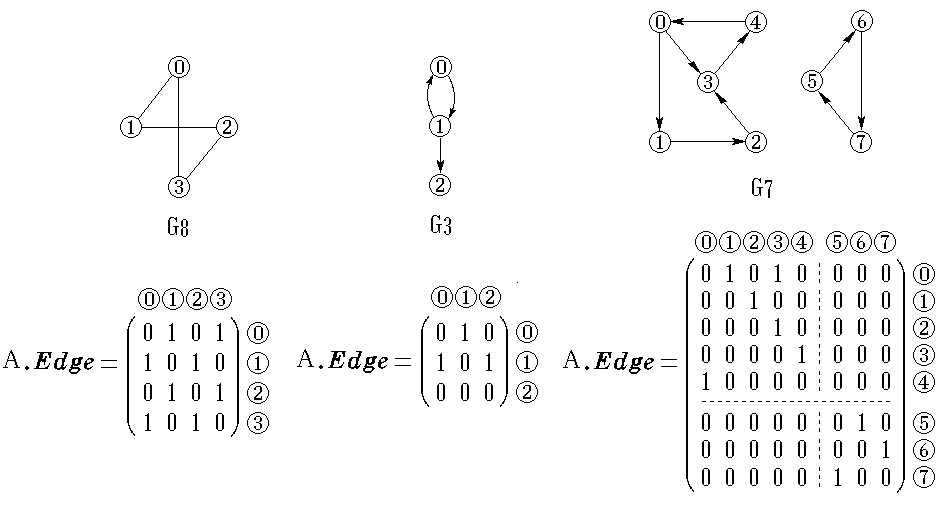
**6.1.2图的存储表示**

**邻接矩阵(Adjacency Matrix)**

* 在图的邻接矩阵表示中，有一个记录各个顶点信息的顶点表，还有一个表示各个顶点之间关系的邻接矩阵。
* 设图A=(V,E)是一个有n个顶点的图，则图的邻接矩阵是一个二维数组A.edge[n][n]，定义：



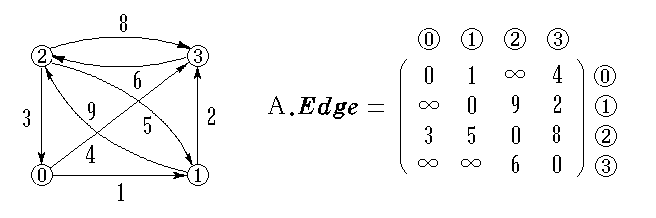
* 无向图的邻接矩阵是对称的，有向图的邻接矩阵可能是不对称的。



* 在有向图中,统计第i行1的个数可得顶点i的出度，统计第j行1的个数可得顶点j的入度。
* 在无向图中,统计第i行(列)1的个数可得顶点i的度。

**网络的邻接矩阵**





**用邻接矩阵表示额图的类的定义**

const int MaxEdges=50;

const int MaxVertices=10;

template<class NameType,class DistType>

class Graph{

private:

SeqList<NameType>VerticesList(MaxVertices);

DistType Edge[MaxVertices][MaxVertices];

int CurrentEdges;

int FindVertex(SeqList<NameType>&L;

const NameType&vertex)

{return L.Find(vertex);}

int GetVertexPos(Const NameType&vertex)

{return FindVertex(VerticesList,vertex);}

public:

Graph(const int sz=MaxNumEdges);

int GraphEmpty()

const{return VerticesList.IsEmpty();}

int GraphFull()

const{return VerticesList.IsFull()||

CurrentEdges==MaxEdges;}

int NumberOfVertices()

{return VerticesList.last;}

int NumberOfEdges(){return CurrentEdges;}

NameType GetValue(const int i)

{return i>=0&&i<VerticesList.last

?VerticesList.data[i]:NULL;}

int GetWeight(const int v1,const int v2);

int GetFirstNeighbor(const int v);

int GetNextNeighbor(const int v1,const int v2);

void InsertVertex(const NameType&vertex);

void InsertEdge

(const int v1,const int v2,DistType weight);

void RemoveVertex(const int v);

void RemoveEdge(const int v1,const int v2);

}

**邻接矩阵实现的部分图操作**

template<class NameType,class DistType>

Graph<NameType,DistType>::Graph(const int sz){

//构造函数

for(int i=0;i<sz;i++)

for(int j=0;j<sz;j++)Edge[i][j]=0;

CurrentEdges=0;

}

template<class NameType,class DistType>

DistType Graph<NameType,DistType>::

GetWeight(const int v1,const int v2){

//给出以顶点v1和v2为两端点的边上的权值

if ( v1 != -1 && v2 != -1 )

return Edge[v1][v2];

else return 0;

}

template<class NameType,class DistType>

int Graph<NameType,DistType>::

GetFirstNeighbor(const int v){

//给出顶点位置为v的第一个邻接顶点的位置

if(v!=-1){

for(int col=0;col<CurrentEdges;col++)

if(Edge[row][col]>0)return col;

}

return-1;

}

template<class NameType,class DistType>

int Graph<NameType,DistType>::

GetNextNeighbor(const int v1,const int v2){

//给出顶点v1的某邻接顶点v2的下一个邻接顶点

if(v1!=-1&&v2!=-1){

for(int col=v2+1;col<CurrentEdges;col++)

if(Edge[v1][col]>0)return col;

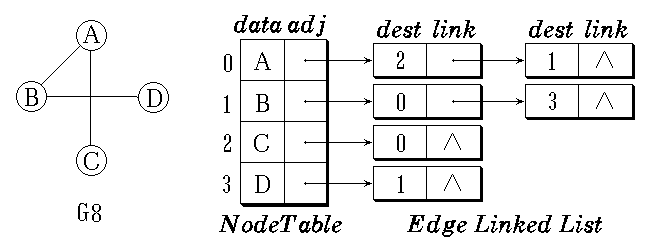
}

return-1;

}

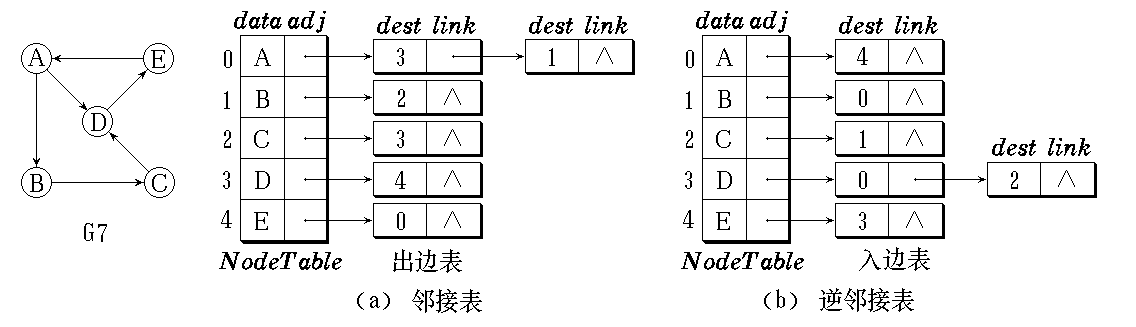
**6.1.3邻接表**

无向图的邻接表



把同一个顶点发出的边链接在同一个边链表中，链表的每一个结点代表一条边，叫做边结点，结点中保存有与该边相关联的另一顶点的顶点下标dest和指向同一链表中下一个边结点的指针link。

**6.14有向图的邻接表和逆邻接表**



* 在有向图的邻接表中，第i个边链表链接的边都是顶点i发出的边。也叫做出边表。
* 在有向图的逆邻接表中，第i个边链表链接的边都是进入顶点i的边。也叫做入边表。
* 带权图的边结点中保存该边上的权值cost。
* 顶点i的边链表的表头指针adj在顶点表的下标为i的顶点记录中，该记录还保存了该顶点的其它信息。
* 在邻接表的边链表中，各个边结点的链入顺序任意，视边结点输入次序而定。
* 设图中有n个顶点，e条边，则用邻接表表示无向图时，需要n个顶点结点，2e个边结点；用邻接表表示有向图时，若不考虑逆邻接表，只需n个顶点结点，e个边结点。