第七章 图

7.1 基本概念和术语

7.2 图的存储结构

7.3 图的遍历

图的应用

7.4 最小代价生成树

7.5 拓扑排序、关键路径

7.6 最短路径

1

图（Graph）是一种较线性表和树更为复杂的数据结构。

- 线性表：数据元素之间构成线性关系，每个数据元素只有一个直接前驱和一个直接后继。
- 树：数据元素之间具有层次关系，每一层上的元素可能和下层多个元素相关，但只能和上层中一个元素相关。
- 图：图结构中，结点之间的关系可以是任意的，任意两个数据元素之间都可能相关。
- 图的应用广泛：
  - ✓ 电路网络分析、交通运输、管理与线路的铺设
  - ✓ 印刷电路板与集成电路的布线、社会网络、WEB链接图
  - ✓ 工程进度的安排、课程表的制订、关系数据库的设计

2

7.1 图的定义和术语

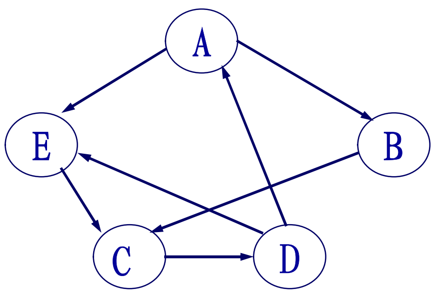
一、图的结构定义:

- 图是由一个顶点集 V 和一个顶点间的关系集合弧集VR（边的集合）构成的数据结构。
  - ✓ 可以用二元组定义为： Graph = (V， VR)
  - ✓ 其中， VR = {<v,w>| v,w∈V 且 P(v,w)}
  - ✓ 谓词 P(v,w) 定义了弧 <v,w>的意义或信息。
- 顶点: 图中的数据元素称为顶点(Vertex),
  - ✓ V是顶点的有穷非空集合;
  - ✓ VR是两个顶点之间的关系的集合。
- 若<v,w>∈VR,则<v,w> 表示从v到w的一条弧（Arc).
  - ✓且称v为弧尾(Tail)或为初始点(Initial node),
  - ✓称w为弧头(Head)或为终端点（Terminal node).

3

由于“弧”是有方向的，因此称由顶点集和弧集构成的图为有向图。弧用尖括号表示。

$G_1 = (V_1, VR_1)$



其中：

V_1={A, B, C, D, E}

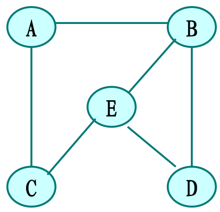
VR_1={<A,B>, <A,E>,<B,C>, <C,D>,<D,B>,<D,A>, <E,C>}

4

若<v,w> ∈VR必有<w,v> ∈VR,则以(v,w) 代替，表示两顶点之间有边（边的顶点对是无序的） 。

此时的图为无向图（Undigraph)。

例：无向图 G2

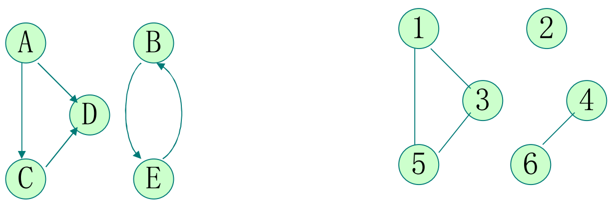


其中:

V2={A, B, C, D, E}

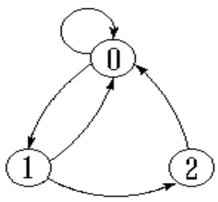
VR2={（A,B）,（A,C),(E,C), (E,D),(D,B) }

有向图、无向图示例

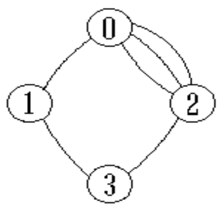


G1 G2

下列形式的图本章不作讨论



（a）带自身环的图



（b）多重图

完全图、稀疏图与稠密图

n:图中顶点的个数；　**e:**图中边或弧的数目。

无向图其边数e的取值范围是**0~n(n-1)/2**。

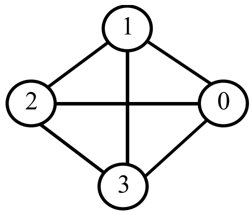
无向完全图：有**n(n-1)/2**条边的无向图。

有向图其边数e的取值范围是**0~n(n-1)**。

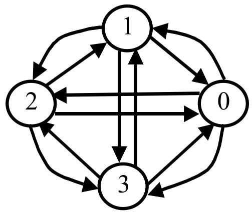
有向完全图：有**n(n-1)**条弧的有向图。

稀疏图：对于有很少条边的图(***e<nlogn***)，

反之称为稠密图。



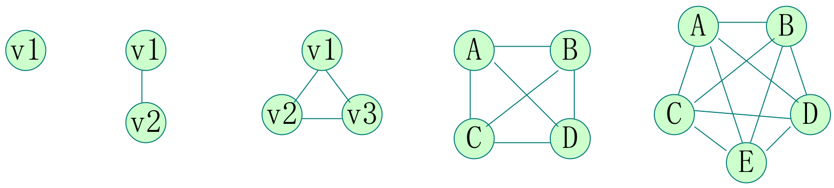
(a)



(b)

7

- 完全图----有n个顶点和n(n-1)/2条边的无向图

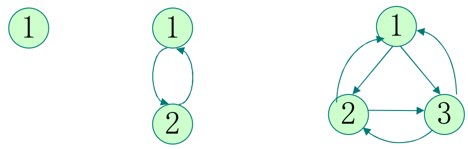


G1	G2	G3	G4	G5
e=1(1-1)/2 =0	e=2(2-1)/2 =1	e=3(3-1)/2 =3	e=4(4-1)/2 =6	e=5(5-1)/2 =10

完全图中任意两个顶点都有一条边相连接。

8

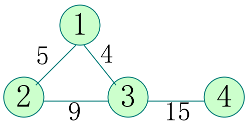
● 有向完全图----有n个顶点和n(n-1)条弧的有向图。



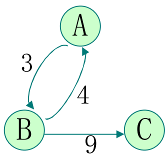
G1　　G2　　G3

e=1(1-1)=0　　e=2(2-1)=2　　e=3(3-1)=6

● 网(Network)----边(弧)上带权(weight)的图。



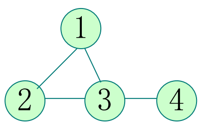
无向网G1



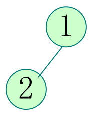
有向网G2

● 有两个图**G=(V, E)**和图**G′=(V′, E′)**,

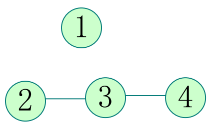
若**V′**⊆**V**且**E′**⊆**E**, 则称图**G′**为**G**的子图。



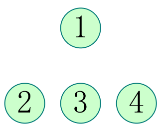
G



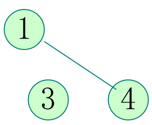
G1



G2



G3



G4

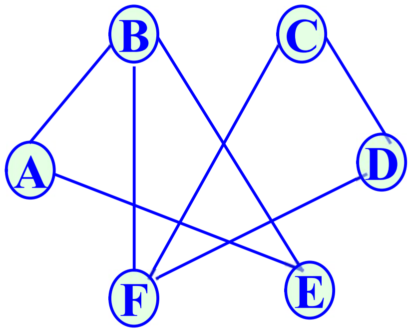
G1,G2,G3是G的子图　G4不是G的子图

无向图的邻接点及关联边

- 假若顶点v 和顶点w 之间存在一条边，则称顶点v 和w 互为邻接点，
- 边(v,w) 和顶点v 和w 相关联。
- 顶点v的度：和顶点v 关联的边的数目，记为TD(V)。

TD(B) = 3

TD(A) = 2



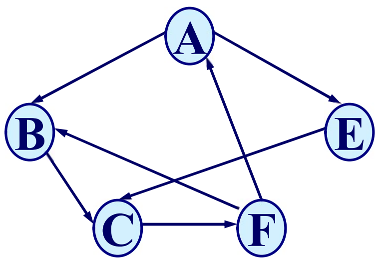
有向图顶点的度、入度、出度

- 顶点V的出度=以V为起点有向边/弧数，记为OD(V)
- 顶点V的入度=以V为终点有向边/弧数，记为ID(V)
- 顶点V的度= V的出度+V的入度，则TD(V)=OD(V)+ID(V)

OD(B) = 1

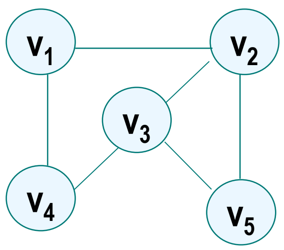
ID(B) = 2

TD(B) = 3



设图G的顶点数为n，边或弧数为e，顶点v_i的度记为$TD(v_i)$，则

$$e = \frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$



G_2

顶点	度
v_1	2
v_2	3
v_3	3
v_4	2
v_5	2

路径、回路

无向图G ＝（V，E）中的顶点序列$v_1, v_2, \ldots, v_k$, 若$(v_i, v_{i+1}) \in E$($i=1, 2, \ldots, k-1$)，$v = v_1$，$u = v_k$,

则称该序列是从顶点v到顶点u的路径；

若v=u，则称该序列为回路；

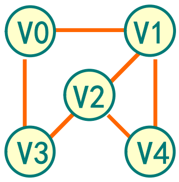
有向图**D** ＝（V，E）中的顶点序列$v_1, v_2, \ldots, v_k$，若$\langle v_i, v_{i+1}\rangle \in E$ ($i=1, 2, \ldots, k-1$)，$v = v_1$，$u = v_k$,

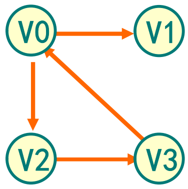
则称该序列是从顶点v到顶点u的路径；

若v=u，则称该序列为回路；

路径长度：路径上的边或弧的数目。

无向图G1





有向图**G2**

在G1中，V0,V1,V2,V3 是V0到V3的路径；

V0,V1,V2,V3,V0是回路；

在G2中，V0,V2,V3 是V0到V3的路径；

V0,V2,V3,V0是回路；

简单路径：路径上各顶点 $v_1, v_2, ..., v_m$ 均不互相重复。

15

Distance and Diameter

- **Distance:** The ***distance*** $\delta_G(u; v)$ from a vertex ***u*** to a vertex ***v*** in a graph G is the shortest path (minimum number of edges) from ***u*** to ***v***.
  - ✓ It is also referred to as the shortest path length from ***u*** to ***v***.
- **Diameter:** The ***diameter*** of a graph is the maximum shortest path length over all pairs of vertices:

$$\mathrm{diam}(G) = \max\ \{\delta_G(u; v)\ : u; v \in V\}$$

16

Some Notations

- For an undirected graph G = (V,E)

$N_G(v) = \{u \mid (u,v) \in E\}$ Neighborhood of a vertex v

$d_G(v) = |N_G(v)|$ Degree of a vertex v

- For a directed graph G = (V,E)

$N_G^+(v) = \{u \mid < v,u > \in E\}$ Set of out-neighbors of a vertex v

$N_G^-(v) = \{u \mid < u,v > \in E\}$ Set of in-neighbors of a vertex v

$N_G^+(U) = \cup_{u \in U} N_G^+(u)$, a set of vertices $U \in V$

$d_G^+(v) = |N_G^+(v)|$ $out-\deg ree$

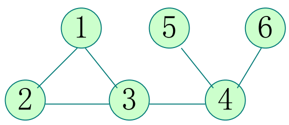
$d_G^-(v) = |N_G^-(v)|$ $in-\deg ree$

$N_G(v)$ $N_G(U)$

17

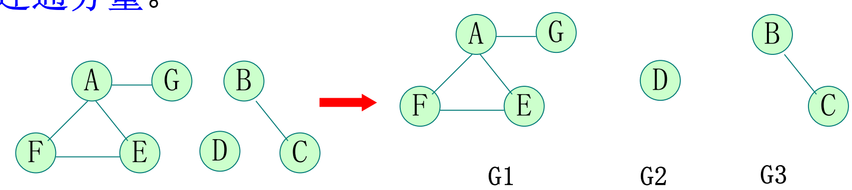
对无向图G:

- 若从顶点v_i到v_j有路径，则称v_i和v_j是连通的。
- 若图G中任意两顶点是连通的，则称G是连通图。



G

- 若图G′是G的一个极大连通子图，则称G′是G的一个连通分量。



G

G有三个连通分量

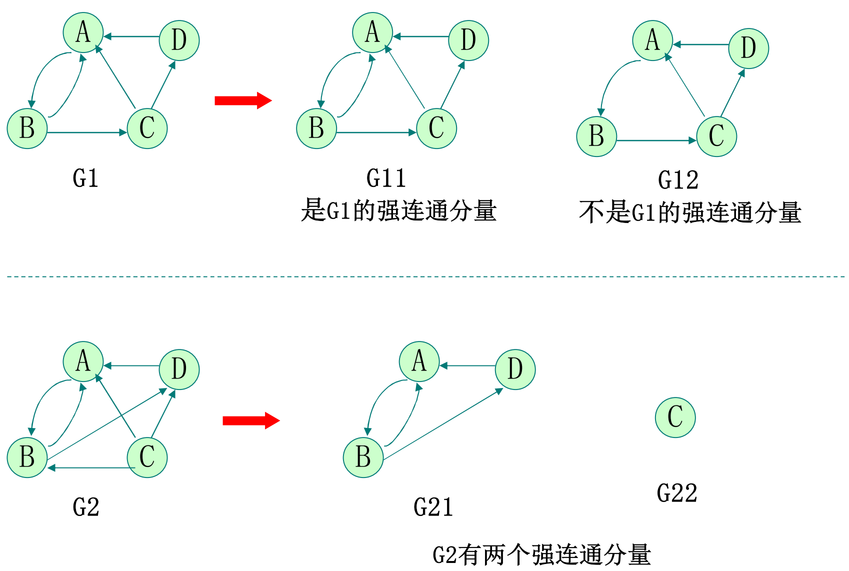
18

对有向图G

- 若在图G中，每对顶点v_i和v_j之间，从v_i到v_j，且从v_j到v_i都存在路径，则称G是强连通图。

- 若图G'是G的一个极大强连通子图，则称G'是G的一个强连通分量。
- 强连通图的强连通分量是自身。

19

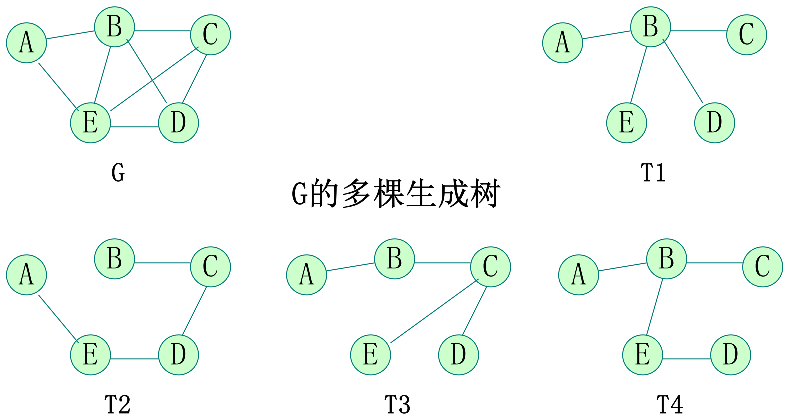


20

●生成树：假设一个连通图有n个顶点和e条边，其中n-1条边和n个顶点构成一个极小连通子图，称该极小连通子图为此连通图的生成树。

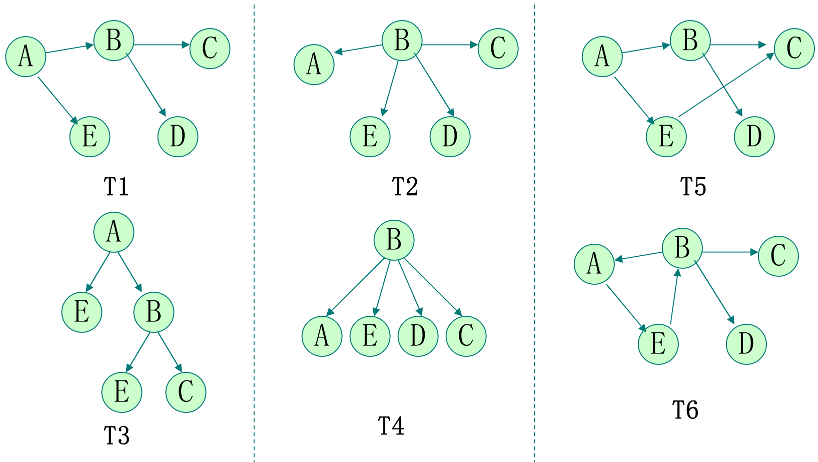
在生成树中添加一条边之后，必定会形成回路或环。

若图中有n个顶点，却少于n-1条边，必为非连通图。



21

● 若有向图G有且仅有一个顶点的入度为0，其余顶点的入度为1，则G是一棵有向树。

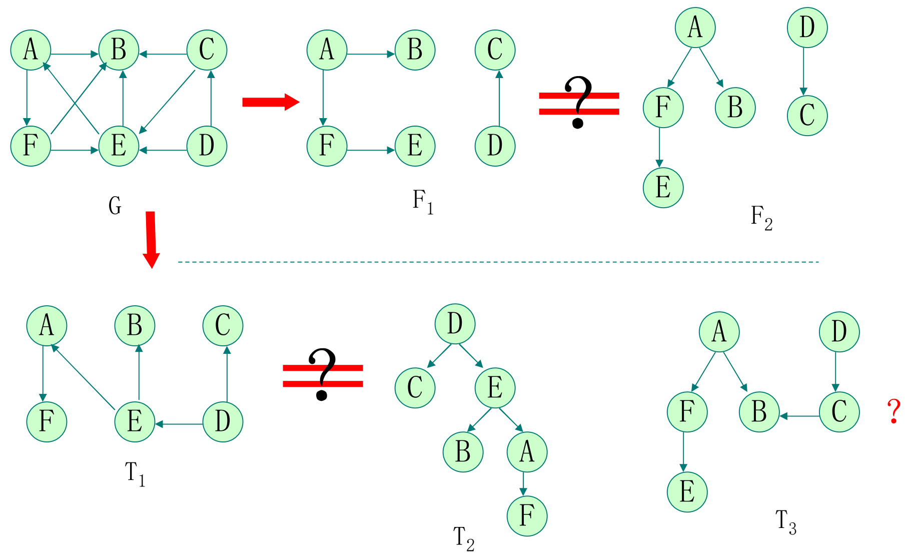


T1，T2，T3，T4是有向树　　T5，T6不是有向树

22

- 有向图的生成树/生成森林：

含有图的全部顶点，但只有足以构成若干互不相交的有向树的弧。



图的抽象数据类型定义

- ADT Graph{

  数据对象**V**：**V**是具有相同特性的数据元素的集合，即顶点集

  数据关系**R**：R＝{VR}

  VR＝{<v,w>|v,w∈V 且P（v,w)}

  基本操作**P**：

  CreateGraph(&G,V,VR)； LovateVex(G,u);

  GetVex(G,v); PutVex(&G,v,value)

  FirstAdjVex(G,v); NextAdjVex(G,v,w);

  DFSTraverse(G,v,visit()); BFSTraverse(G,v,visit());

- }ADT Graph

基本操作

结构的建立和销毁

插入或删除顶点

对邻接点的操作

对顶点的访问操作

插入和删除弧

遍历

对顶点的访问操作

LocateVex(G, u);

// 若G中存在顶点u，则返回该顶点在

// 图中 “位置” ； 否则返回其它信息。

GetVex(G, v); // 返回 v 的值。

PutVex(&G, v, value);

// 对 v 赋值value。

对邻接点的操作

FirstAdjVex(G, v);

// 返回 v 的"第一个邻接点"。若该顶点

//在 G 中没有邻接点，则返回"空"。

NextAdjVex(G, v, w);

// 返回 v 的（相对于 w 的）"下一个邻接点"。

//若 w 是 v 的最后一个邻接点，则返回"空"。

27

遍 历

DFSTraverse(G, v, Visit());

//从顶点v起深度优先遍历图G，并对每

//个顶点调用函数Visit一次且仅一次。

BFSTraverse(G, v, Visit());

//从顶点v起广度优先遍历图G，并对每

//个顶点调用函数Visit一次且仅一次。

28

7.2 图的存储结构

一、图的数组(邻接矩阵)存储表示

二、图的邻接表存储表示

三、有向图的十字链表存储表示

四、无向图的邻接多重表存储表示

五、面向并行处理的图表示

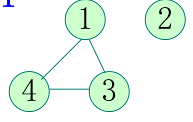
7.2.1 数组表示法/邻接矩阵

顶点数组---用一维数组存储顶点(元素)

邻接矩阵---用二维数组存储顶点(元素)之间的关系(边或弧)

$$A[i][j]=\begin{cases}1 & \text{顶点}v_i\text{与}v_j\text{间有边(弧)}\\ 0 & \text{顶点}v_i\text{与}v_j\text{间无边(弧)}\end{cases}$$

例1



G

V[1..4]

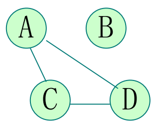
1	2	3	4
1	2	3	4

顶点数组

$$M=\begin{array}{c} \\ 1\\ 2\\ 3\\ 4\end{array}\begin{array}{c}\begin{array}{cccc}1&2&3&4\end{array}\\ \begin{pmatrix}0&0&1&1\\0&0&0&0\\1&0&0&1\\1&0&1&0\end{pmatrix}\end{array}$$

邻接矩阵

例2 无向图G



G

V[1..4]

A	B	C	D
1	2	3	4

顶点数组

$$M=\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 0 & 1 & 1 \\ B & 0 & 0 & 0 & 0 \\ C & 1 & 0 & 0 & 1 \\ D & 1 & 0 & 1 & 0 \end{array}$$

邻接矩阵

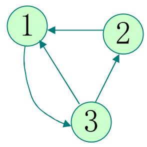
顶点v_i的 TD(v_i)=M中第i行元素之和

$$=\sum_{j=1}^{n} M[i][j]$$

顶点v_i的 TD(v_i)=M中第i列元素之和

$$=\sum_{j=1}^{n} M[j][i]$$

例3 求下列有向图的邻接矩阵



$$M=\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \end{array}$$

邻接矩阵

顶点v_j的 ID(v_j)=M中第j列元素之和

$$=\sum_{i=1}^{n} M[i][j]$$

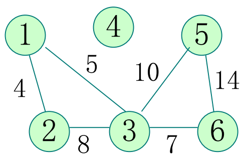
顶点v_i的 OD(v_i)=M中第i行元素之和

$$=\sum_{j=1}^{n} M[i][j]$$

网的邻接矩阵:若图**G**是一个有**n**个顶点的网，则它的邻接矩阵是具有如下性质的**n**×**n**矩阵**A**：

$$A[i,j]=\begin{cases} w_{ij}, & \text{若}(v_i,v_j)\text{或}<v_i,v_j>\in V \\ \infty, & \text{其它} \end{cases}$$

例4



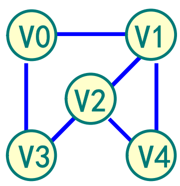
网G

$$M=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \left(\begin{array}{cccccc} \infty & 4 & 5 & \infty & \infty & \infty \\ 4 & \infty & 8 & \infty & \infty & \infty \\ 5 & 8 & \infty & \infty & 10 & 7 \\ \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 10 & \infty & \infty & 14 \\ \infty & \infty & 7 & \infty & 14 & \infty \end{array}\right) \end{array}$$

邻接矩阵

数组表示法的特点（无向图）

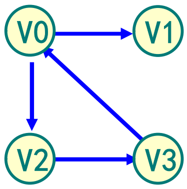
- ❑无向图的邻接矩阵是对称矩阵，同一条边表示了两次；
- ❑顶点**v**的度：等于二维数组对应行（或列）中值为**1**的元素个数；
- ❑判断两顶点v、u是否为邻接点：
  - ✓只需判二维数组对应分量是否为1；
- ❑顶点不变，在图中增加、删除边：
  - ✓只需对二维数组对应分量赋值1或0；
- ❑设图的顶点数为 n ，用有n个元素的一维数组存储图的顶点，用邻接矩阵表示边，则G占用的存储空间为：$n+n^2$；图的存储空间占用量只与它的顶点数有关，与边数无关；适用于边稠密的图；



$$\Rightarrow \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

数组表示法的特点（有向图）

- 有向图的邻接矩阵不一定是对称的；
- 顶点v的出度：等于二维数组对应行中值为1的元素个数；
- 顶点v的入度：等于二维数组对应列中值为1的元素个数；



$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

邻接矩阵法优点：容易实现图的操作，

如：求某顶点的度、找顶点的邻接点等等。仅耗费 O(n) 时间。

邻接矩阵法缺点：n个顶点需要n^2 个单元存储边(弧)。

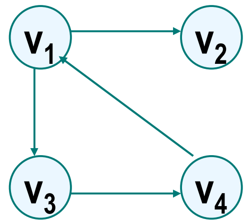
35

邻接矩阵表示法类型描述

```
#define MAX_VERTEX_NUM  20      //最多顶点个数
#define INFINITY  INT_MAX       //表示极大值∞
typedef enum{DG, DN, UDG, UDN} GraphKind;
typedef struct ArcCell{
    VRType   adj;    //顶点关系类型，无权图取1或0；有权图取权值类型
    InfoType  *info;              //该弧相关信息的指针
}ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
typedef struct{
    VertexType vexs [MAX_VERTEX_NUM]; //顶点向量
    AdjMatrix  arcs;  //邻接矩阵n*n
    int  vexnum,  arcnum; //图的顶点数n和弧数e
    GraphKind kind;  //图的种类标志
} MGraph;
```

36

示例



G_1

G1. vexs[ ]={"v_1", " v_2", " v_3", " v_4"}

$$G1.arcs[\,][\,]=\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

G1.vexnum=4 G1.arcnum=4 G1.kind=DG

37

```
Status CreateUDN(Mgraph &G){
        //无向网的构造，用邻接矩阵表示,算法7.2
  scanf(&G.vexnum, &G.arcnum, &IncInfo); //输入顶点数n弧数e和信息
  for(i=0;i<G.vexnum;++i) scanf(&G.vexs[i] );          //构造顶点向量
  for(i=0; i<G.vexnum; ++i)                 //对邻接矩阵n*n个单元初始化
     for(j=0;j<G.vexnum;++j) G.arcs[i][j]={INFINITY, NULL};
  for(k=0;k<G.arcnum;++k){
     //给邻接矩阵有关单元赋初值(循环次数=弧数e)
     scanf(&v1, &v2, &w);                 //输入弧的两顶点以及对应权值
     i=LocateVex(G,v1); j=LocateVex(G,v2); //找到两顶点在G中的位置
     G.arcs[i][j].adj=w; //输入对应权值
     If(IncInfo) Input(*G.arcs[i][j].info);          //如果弧有信息则填入
     G.arcs[j][i]=G.arcs[i][j] ;                     //无向网是对称矩阵
  }
  return OK;
  } // CreateUDN
```

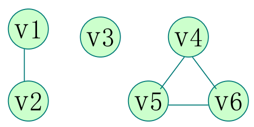
对于n个顶点e条弧的网，
建网时间效率 = $O(n+n^2+e*n)$

38

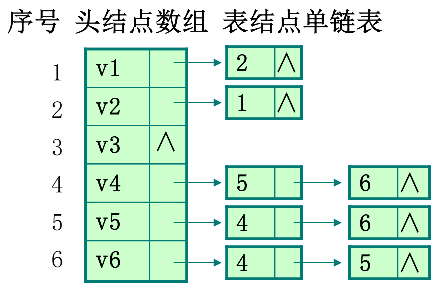
7.2.2 邻接表、逆邻接表 ： 链式存储结构。

（1）无向图的邻接表:

为图G的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v_i的边。



图G



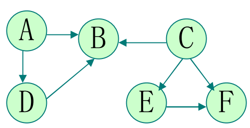
图G邻接表

- 在G的邻接表中，同一条边对应两个结点。
- 无向图G的邻接表，顶点v_i的度为第i个单链表的长度。
- 若无向图G有n个顶点和e条边，需n个表头结点和2e个表结点。

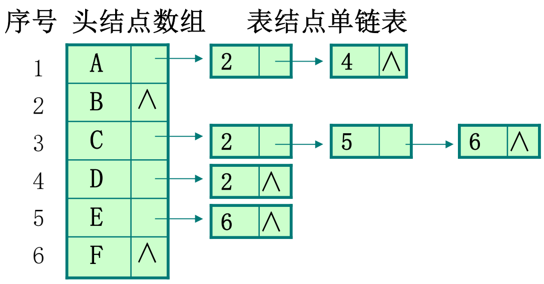
39

（2）有向图的邻接表 ：

第i个单链表中的表结点，表示以顶点v_i为尾的弧(v_i, v_j)的弧头。



有向图G

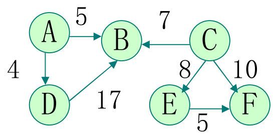


图G的邻接表

- 若有向图G有n个顶点和e条弧，则需n个表头结点和e个表结点。
- 有向图G的邻接表，顶点v_i的出度为第i个单链表的长度。
- 求顶点v_i的入度需遍历全部单链表，统计结点值为i的结点数。

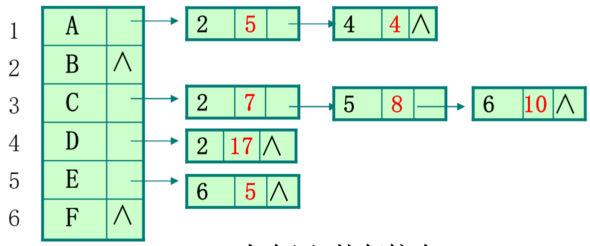
40

（3）有向网的邻接表



有向网G

序号 头结点数组 表结点单链表

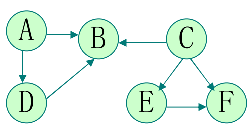


有向网G的邻接表

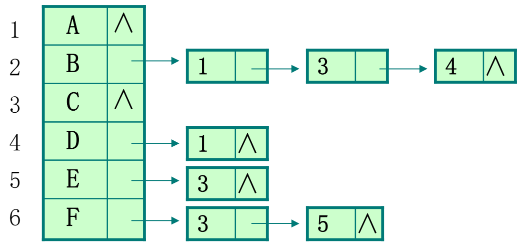
（4）有向图的逆邻接表

第i个单链表中的表结点，表示以顶点v_i为头的弧(v_j, v_i)的弧尾。

序号 头结点数组 表结点单链表



有向图G



图G的逆邻接表

- 若有向图G有n个顶点e条弧，则需n个表头结点和e个表结点。
- 有向图G的逆邻接表，顶点v_i的入度为第i个单链表的长度。
- 求顶点v_i的出度需遍历全部单链表，统计结点值为i的结点数。

邻接表的优点：

空间效率高；容易寻找顶点的邻接点；

邻接表的缺点：

判断任意两顶点间是否有边或弧，需搜索两结点对应的单链表，没有邻接矩阵方便。

邻接表的结点结构：

表结点

adjvex	info	nextarc
指示与顶点V_i邻接的点在图中的位置	信息	指示下一条边或弧的结点

头结点

data	firstarc
存储顶点Vi的名或其他信息	指示链表中第一个结点

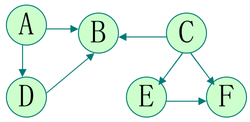
43

邻接表存储结构的类型定义：

```
#define MAX_VERTEX_NUM 20
typedef struct ArcNode{//表结点结构类型
    int adjvex;                  //该弧(边)的终点位置
    struct ArcNode *nextarc;     //指向下一条弧的指针
    InfoType *info;              //该弧的相关信息的指针
} ArcNode;
typedef struct Vnode {//头结点的类型
    Vertex data;                 //顶点信息
    ArcNode *firstarc;           //指向第一条弧
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {//邻接表
    AdjList vertices;
    int vexnum, arcnum;          //图中顶点数n和边数e
    int kind;                    //图的类型
} ALGraph;
```

44

```
typedef  struct{//邻接矩阵
    VertexType  vexs [MAX_VERTEX_NUM];
    AdjMatrix   arcs;  //邻接矩阵n*n
    int   vexnum,   arcnum;
    GraphKind  kind;
} MGraph;
typedef struct {//邻接表
    AdjList vertices;
    int vexnum, arcnum;
    int kind;
} ALGraph;
```

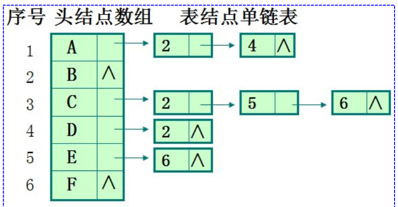


有向图G

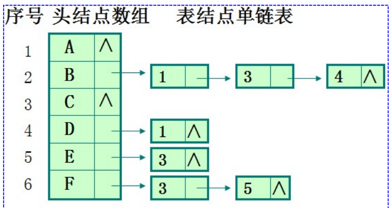
顶点数组

序号	顶点		1	2	3	4	5	6
1	A	1	0	1	0	1	0	0
2	B	2	0	0	0	0	0	0
3	C	3	0	1	0	0	1	1
4	D	4	0	1	0	0	0	0
5	E	5	0	0	0	0	0	1
6	F	6	0	0	0	0	0	0

图G的邻接矩阵



图G的邻接表



图G的逆邻接表

讨论：邻接表与邻接矩阵有什么异同之处？

1. 联系：

邻接表中每个链表对应于邻接矩阵中的一行；

链表中结点个数等于邻接矩阵一行中非零元素的个数。

2. 区别：

① 对于任一确定的无向图，邻接矩阵是唯一的（行列号与顶点编号一致）；

但邻接表不唯一（链接次序与顶点编号无关）。

② 邻接矩阵的空间复杂度为$O(n^2)$；

而邻接表的空间复杂度为$O(n+e)$。

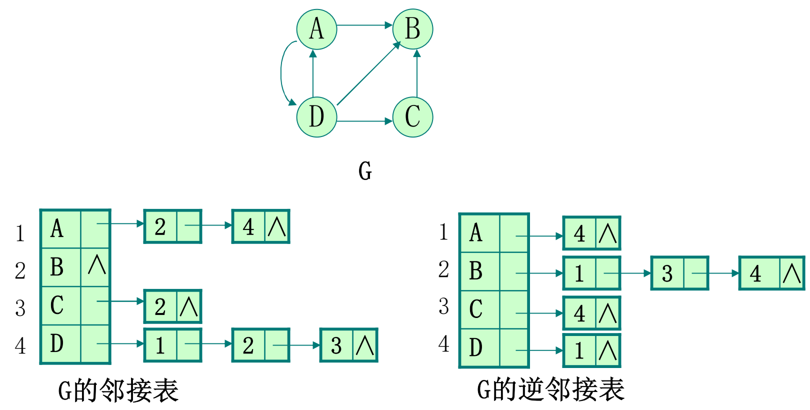
3. 用途：

邻接矩阵多用于稠密图的存储（e接近n(n-1)/2）；

而邻接表多用于稀疏图的存储（$e<<n^2$）

7.2.3 有向图的十字链表

将邻接表和逆邻接表合并而成的链接表。



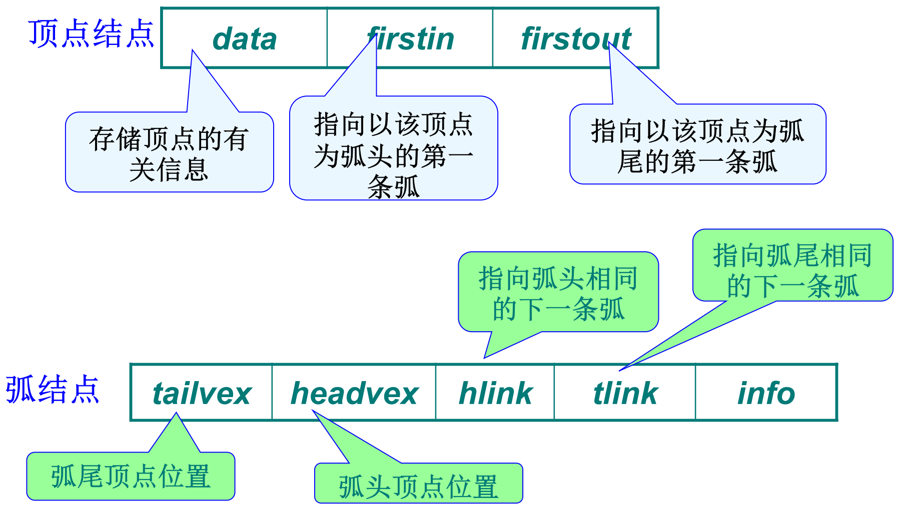
➢以邻接表为基础，扩展结点属性成起止结点序号

➢再添加逆邻接表信息

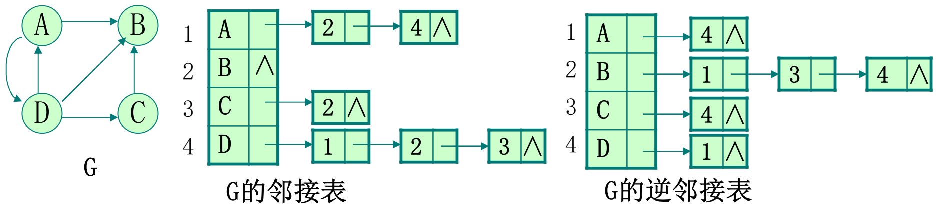
47

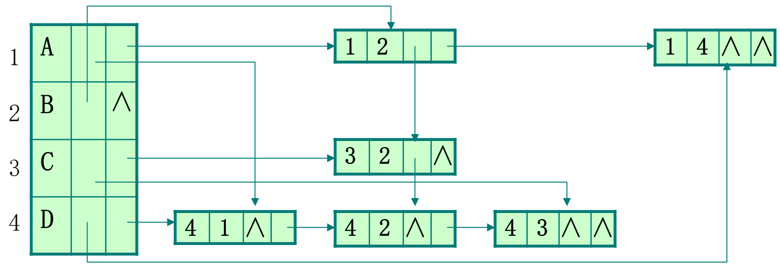
1.十字链表 是有向图的另一种链式存储结构

2.结点结构设计



48





G的十字链表

① 顶点结点数=顶点数　　弧结点数=弧的条数

② 求入度：从顶点V_i的firstin出发，沿着弧结点中的hlink所经过的弧结点数。

求出度：从顶点V_i的firstout出发，沿着弧结点中的tlink所经过的弧结点数。

49

有向图的十字链表存储表示

弧结点结构

tailvex	headvex	hlink	tlink	
弧尾顶点位置	弧头顶点位置			弧的相关信息

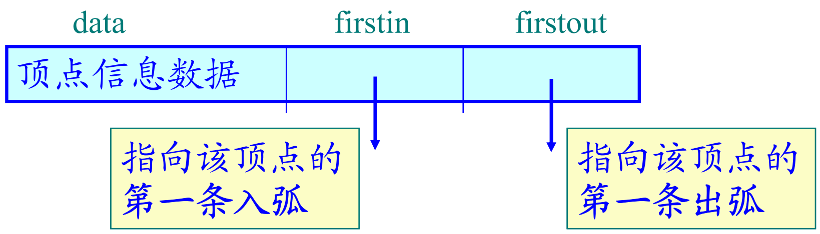
hlink：指向下一个有相同弧头的结点

tlink：指向下一个有相同弧尾的结点

```
typedef struct ArcBox { // 弧的结构表示
    int tailvex, headvex;
    InfoType  *info;
    struct ArcBox  *hlink, *tlink;
    } ArcBox ;
```

50

顶点结点结构




```
typedef struct VexNode { // 顶点的结构表示
    VertexType  data;
    ArcBox  *firstin, *firstout;
} VexNode;
```

51

有向图的结构表示(十字链表)

```
typedef struct {
   VexNode  xlist[MAX_VERTEX_NUM];
           // 顶点结点(表头数组)
   int   vexnum, arcnum;
           //有向图的当前顶点数和弧数
} OLGraph;
```

十字链表优点：易于实现操作，如求顶点的入度、出度等。

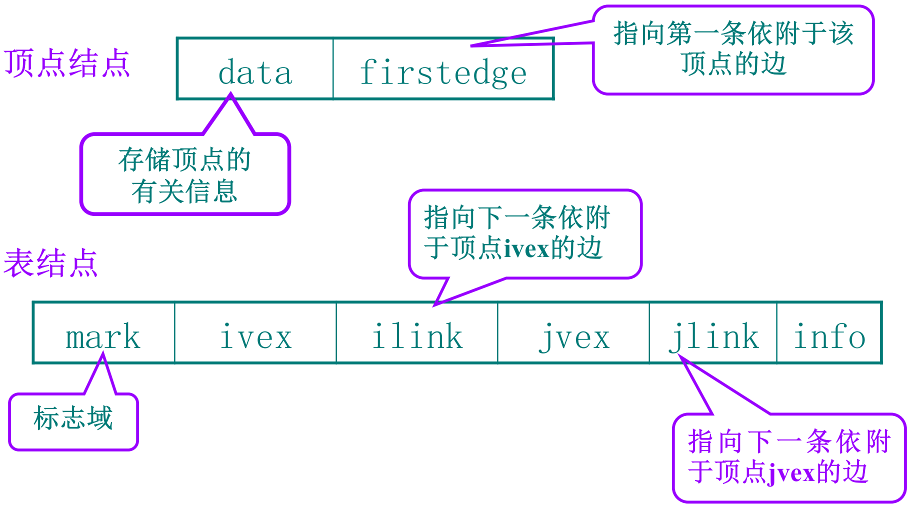
空间复杂度和建表的时间复杂度都与邻接表相同。

52

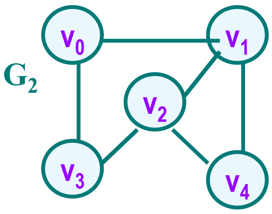
7.2.4 邻接多重表

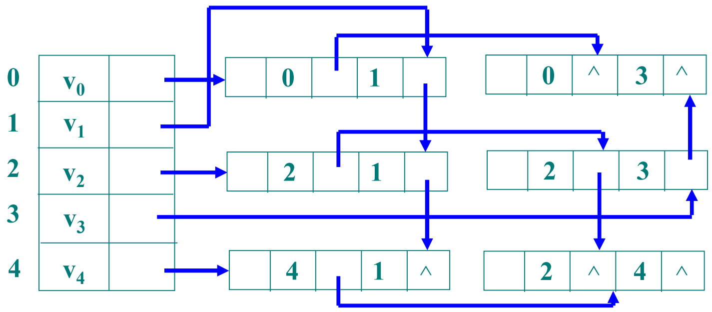
（无向图的）的另一种链式存储结构

1.结点结构设计



2. 邻接多重链表的示例及分析





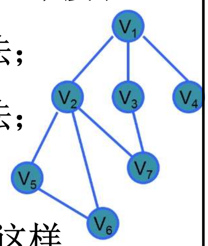
每个链表结点处在两个链表中。

例 给定一个具有n个结点的无向图的邻接矩阵和邻接表。

（1） 设计一个将邻接矩阵转换为邻接表的算法；

（2） 设计一个将邻接表转换为邻接矩阵的算法；

（3） 分析上述两个算法的时间复杂度。

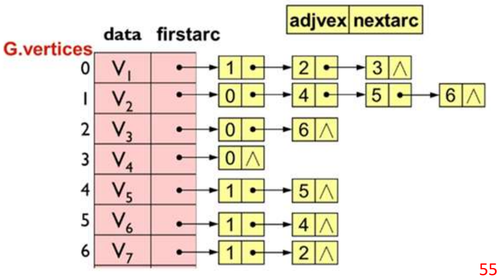


解：（1）在邻接矩阵上查找值不为0的元素，找到这样的元素后创建一个表结点并在邻接表对应的单链表中采用前插法插入该结点。算法如后：

G.vexs=

0	1	2	3	4	5	6
V_1	V_2	V_3	V_4	V_5	V_6	V_7

$$
G.arcs=\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$



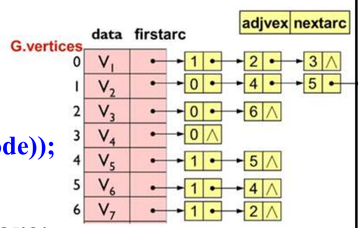
55

```
void MatToList(MGraph g, ALGraph *&G)
/*将邻接矩阵g转换成邻接表G*/
{  int i,j,n=g.vexnum; ArcNode *p;
   G=(ALGraph *)malloc(sizeof(ALGraph));
   for (i=0;i<n;i++)  //头结点赋值
     { G->vertices[i].data=g.vex[i].data;
       G->vertices[i].firstarc=NULL; }
   for (i=0;i<n;i++)  //检查邻接矩阵每个元素
     for (j=n-1;j>=0;j--)  //采用前插法
       if (g.arcs[i][j]!=0) {
         p=(ArcNode *)malloc(sizeof(ArcNode));
         p->adjvex=j;
         p->nextarc=G-> vertices[i].firstarc;
         G->vertices[i].firstarc=p; }
   G->vexnum=n; G->arcnum=g.arcnum; G->kind=g.kind;
}
```


G.vexs=

0	1	2	3	4	5	6
V_1	V_2	V_3	V_4	V_5	V_6	V_7

$$
G.arcs=\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$



56

```
(2) void ListToMat(ALGraph G, MGraph &g)
{   int i,j,n=G.vexnum; ArcNode *p;
    for (i=0;i<n;i++)                    /*g.arcs[i][j]赋初值0*/
        for (j=0;j<n;j++)  g.arcs[i][j]=0;
    for (i=0;i<n;i++) {
        g.vex[i].data =G.vertices[i].data;
        p=G.vertices[i].firstarc;
         while (p!=NULL) {
             g.arcs[i][p->adjvex]=1;
             p=p->nextarc; }
     }
     g.vexnum=G.vexnum; g.arcnum=G.arcnum; g.kind =G.kind;
}
```

57

(3)上述两个算法的时间复杂度均为$O(n^2)$。

对于(2)的算法,若不计算给a[i][j]赋初值0的双重for循环语句,其时间复杂度为$O(n+e)$,其中e为图的边数。

58

7.2.5 Representing graph for parallel algorithms

- Represent graph based on sets and tables.
- A more abstract view of the representation of graphs and instead of jumping right down to the low-level data structures such as arrays or linked-lists.
- View the standard representations as the special cases when using particular implementations of sets and tables.
- Easier to apply parallel operations to the graphs.
- Edge sets and Adjacency tables.

59

Edge sets

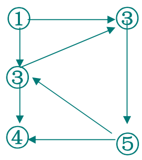
- Represent a graph based on its definition as

  a set of vertices V and a set of directed edges $A \subseteq V \times V$.
- If we use the set ADT, the keys for the edge set are simply pairs of vertices.
- The representation abstracts away from the particular data structure used for the set—the set could be implemented as a list, an array, a tree, or a hash table.

60

练习：

1. 对如下有向图：



试写出：（1）邻接矩阵

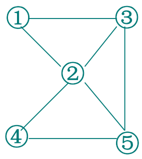
（2）邻接表

（3）逆邻接表

（4）十字链表

（5）并行性表示

2. 对如下无向图：



试写出：（1）邻接矩阵

（2）邻接表

（3）邻接多重表

（4）从顶点1出发按深度和广度优先搜索遍历图的顶点序列

61

7.3　图的遍历

- 从图的某个顶点出发，访问图中的所有顶点，且使每个顶点仅被访问一次的过程叫做图的遍历。
- 遍历方法：深度优先遍历和广度优先遍历
- 图的遍历操作是求解图的连通性问题、拓扑排序等问题的基础。

62

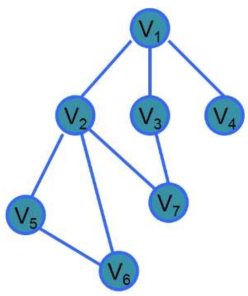
图的遍历操作的复杂性主要表现在以下四个方面：

- 图中顶点的地位都相同，如何确定首结点？
- 非连通图可能有多个连通分量，从一个顶点出发只能访问它所在连通分量上的所有结点，如何访问图中其余的连通分量？
- 在图结构中，如果有回路存在，则一个顶点被访问之后，有可能沿回路又回到该顶点，遍历中访问不能重复；
- 在图结构中，一个顶点可能有多个邻接点，这样当这个顶点访问过后，如何选取下一个要访问的顶点？

63

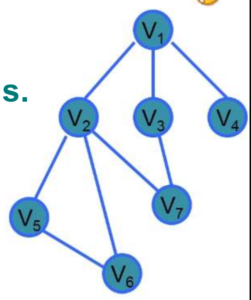
Frontier

- A set of vertices consists of vertices that are connected via an edge to visited vertices.
- A vertex v is in the frontier if v is not yet visited but it is connected via one edge (u, v) to a visited vertex u.
- How to determine the next vertex to jump to in order to traverse paths?
  - ✓ visit only the vertices in the frontier.



64

Algorithm Graph Search



Given: a graph G = (V,E) and a start vertex s.

Frontier F = {s}.

Visited set X = ∅.

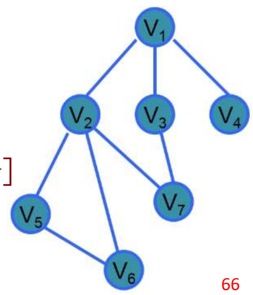
While the frontier is not empty

- **Pick a set of vertices U in the frontier and visit them.**
- **Update the visited set X = X ∪ U.**
- **Extend F with the out-edges of vertices in U.**
- **Delete visited vertices from F, F = F \ X.**

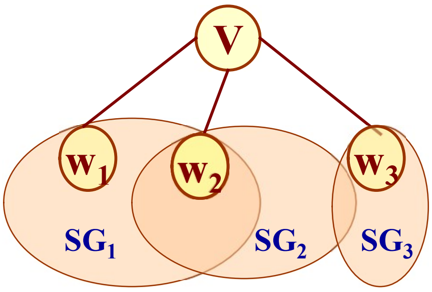
65

7.3.1 图的深度优先搜索 DFS---Depth First Search

- 从图中某顶点v出发：
  - (1)访问顶点v；
  - (2)依次从v的未被访问的邻接点w出发，对图进行深度优先遍历；直至图中和v有路径相通的顶点都被访问；
  - (3)若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。
- 如何判别v的邻接点是否被访问？
  - 为每个顶点设立一个 访问标志 visited[w]



66



W_1、W_2和W_3 均为 V 的邻接点，SG_1、SG_2 和 SG_3 分别为含顶点 W_1、W_2和W_3 的子图。

访问顶点 **V** ；

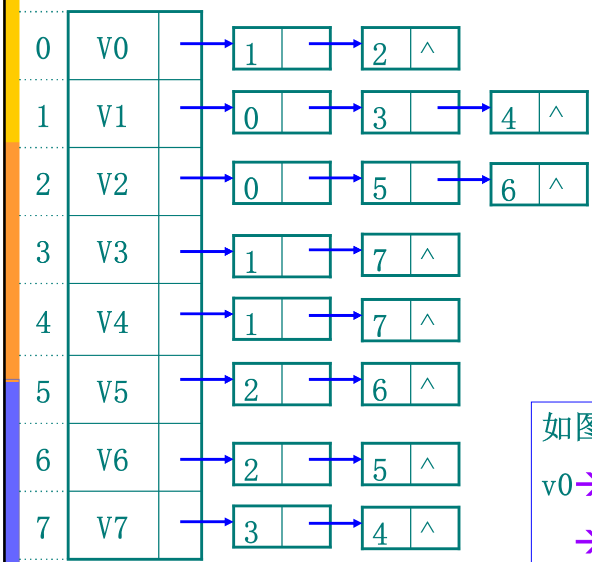
for (W_1、W_2、W_3)

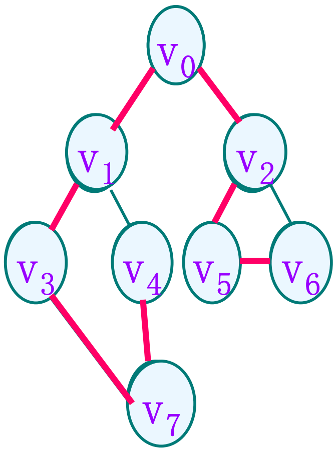
若该邻接点W_i未被访问，

则从它出发进行深度优先遍历。

- 类似于树的先根遍历；
- 由于没有规定访问邻接点的顺序，深度优先序列不唯一
- 可用递归实现

示例及分析1





如图的深度优先搜索序列为：

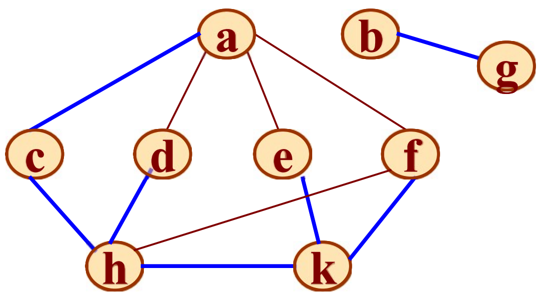
v0→v1→v3→v7→v4

→v2→v5→v6

深度优先搜索算法：

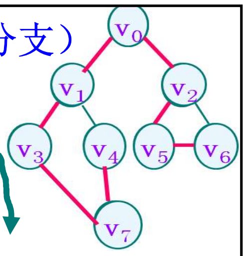
```
Boolean visited[MAX];          /*访问标志数组*/
void  DFSTraverse(Graph G){  //对图 G 作深度优先遍历
      for (v=0;v<G.vexnum;v++) visited[v]=FALSE;
      for (v=0;v<G.vexnum;v++)
          if (!visited[v]) DFS(G,v); //从v0开始
}
void DFS(Graph G,int v){ //从v出发递归地深度优先遍历图G
   visited[v]=TRUE;visit(v);
   for (w=FirstAdjVex(G,v);w>=0;w=NextAdjVex(G,v,w))
   if (!visited[w]) DFS(G,w);     //递归调用DFS
}
```


例如：

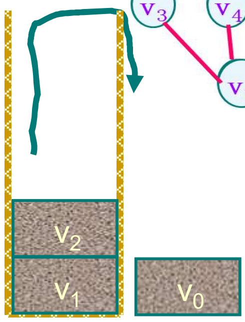


	0	1	2	3	4	5	6	7	8
	a	b	c	d	e	f	g	h	k
访问标志：	T	T	T	T	T	T	T	T	T
访问次序：	a	c	h	d	k	f	e	b	g

用非递归过程实现深度优先搜索（对连通分支）




```
void  DepthFirstSearch(Graph G,  int v0)
{     //从v0出发
  InitStack(S);   //初始化空栈
  Push(S,  v0);
  while ( ! StackEmpty(S)) {
    v=Pop(S);
     if (!visited(v)){ //栈中可能有重复顶点
         visited[v]=True; Visit(v);
     }
     for(w=FirstAdjVex(G,  v); w>=0; w=NextAdjVex (g, v, w)){
           if (!visited[w])  Push(S,  w);
     }
  }
}
```



•顶点先进栈，出栈时若未被访问，则访问它；

•访问之后，其未被访问的邻接点依次进栈。

71

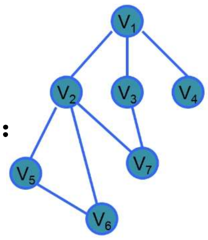
算法分析

- DFS算法是难以并行的处理过程：work = span
- 上述算法对各种存储结构均适用，但其中使用的函数 FirstAdjVex 和NextAdjVex须据特定的存储结构来实现。
- 算法的时间复杂度依赖于所采用的存储结构
  - ❖ 图中有 **n** 个顶点，**e** 条边
  - ❖ 当采用邻接矩阵存储时，时间复杂度为**O**（**n²**）
  - ❖ 当采用邻接表存储时，时间复杂度为**O**（**n+e**）
  - ❖ **// if using the tree-based cost specification for sets runs in O(elog n) work and span. //**

72

7.3.2 图的广度优先搜索BFS---Breadth First Search

- 广度优先搜索遍历类似于树的按层次遍历。
- 设图G的初态所有顶点均未访问，在G 中任选一顶点v 为初始点，则广度优先搜索的基本思想是：



  ①访问顶点v；

  ②访问v 的所有未被访问的邻接点w_1，w_2，…w_k；

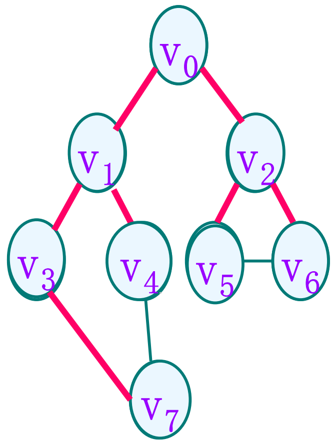
  ③依次从这些邻接点（在步骤②中访问的顶点）出发， 访问它们的所有未被访问的邻接点；依此类推，直到由v可以到达的所有顶点都被访问过为止；

  ④若此时图中还有顶点未被访问，则另选一个未被访问过的顶点作起始点，重复上述过程直至图中所有顶点均被访问到。
- 为实现③，需保存在步骤②中访问的顶点，且访问这些顶点的邻接点的顺序为：先保存的顶点，其邻接点先被访问。

73

示例及分析

0	V0	1 →	2 ^	
1	V1	0 →	3 →	4 ^
2	V2	0 →	5 →	6 ^
3	V3	1 →	7 ^	
4	V4	1 →	7 ^	
5	V5	2 →	6 ^	
6	V6	2 →	5 ^	
7	V7	3 →	4 ^	



如图的广度优先搜索序列为：

v0→v1→v2→

v3→v4→v5→v6→v7

74

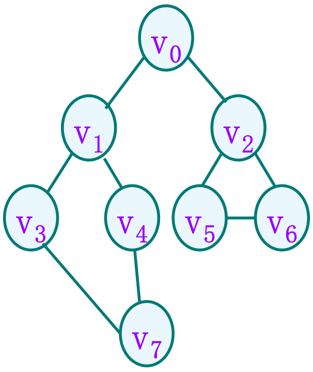
注意：

访问后就入队；

出队后，考察邻接点：

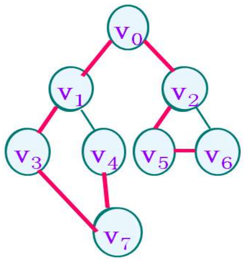
存在吗？

已被访问吗？

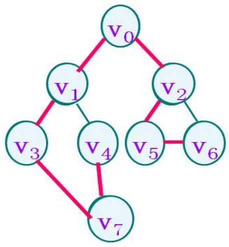


v0							
	v1	v2					
		v2	V3	v4			
			v3	v4	v5	v6	
				v4	v5	v6	v7
					v5	v6	v7
						v6	v7
							v7


```
void BFSTraverse(Graph G,Status (*Visit)(int v)){
//按广度优先非递归遍历图G，使用辅助队列Q和访问标志数组visited
  for (v=0; v<G.vexnum; ++v)
     visited[v] = FALSE;    //初始化访问标志
  InitQueue(Q);          //置空的辅助队列Q
  for ( v=0;  v<G.vexnum;  ++v )
    if ( !visited[v]) {          //v 尚未访问
        … …
    } // if
} // BFSTraverse
```



```
visited[v] = TRUE;  Visit(v);    //访问v
EnQueue(Q, v);              //v被访问后入队列
while (!QueueEmpty(Q))  {
   DeQueue(Q, u);          //队头元素出队并置为u
   for(w=FirstAdjVex(G, u); w>=0; w=NextAdjVex(G,u,w))
      if ( ! visited[w])  {
         visited[w]=TRUE;  Visit(w);
         EnQueue(Q, w);  //访问的顶点w入队列
      } // if
} // while
```



- 顶点未被访问，先访问，再入队；
- 出队之后，依次访问其未被访问的邻接点并入队。
- 队列中顶点皆已被访问。

77

BFS 算法效率分析:

（设图中有 n 个顶点，e 条边）

- 如果使用邻接表来表示图，则BFS循环的总时间代价为 $d_0 + d_1 + \ldots + d_{n-1} = O(e)$，其中的 d_i 是顶点i 的度。

  //BFS算法的时间复杂度为O(n+e)

- 如果使用邻接矩阵，则BFS对每一个被访问到的顶点，都要循环检测矩阵中的一行（n个元素），总的时间代价为$O(n^2)$。

DFS与BFS之比较:

- 空间复杂度相同，都是O(n)(借用堆栈或队列暂存n个顶点)
- 时间复杂度只与存储结构（邻接矩阵或邻接表）有关，而与搜索路径无关。

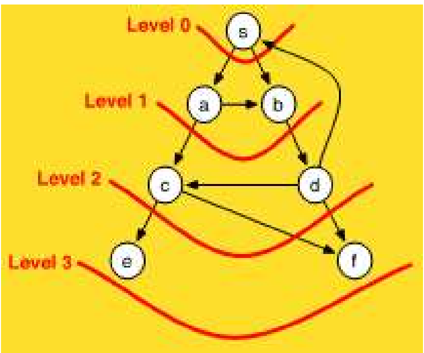
78

BFS Algorithm and Parallelism

- **The idea:** start at a source vertex s and explore the graph outward in all directions level by level,
  - ✓ first visiting all vertices that are the (out-)neighbors of s (i.e. have distance 1 from s),
  - ✓ then vertices that have distance two from s,
  - ✓ then distance three, etc.
- Given a graph G and a source s. The level of a vertex v is defined as the shortest distance from s to v, that is the number of edges on the shortest path connecting s to v.

79

A graph and its levels



80

Algorithm BFS

Given: a graph G = (V,E) and a start vertex s.

Frontier F = {s}.

Visited set X = ∅.

i = 0. // Current level

While F ≠ ∅ do

Visit (F). // visit all the vertices in the frontier

X = X ∪ F. // Update the visited set

F = N_G(F).

// Set F to be the out-neighbors of vertices in F

F = F \ X. // Delete visited vertices

i = i+1. // Next level

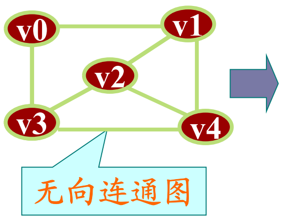
81

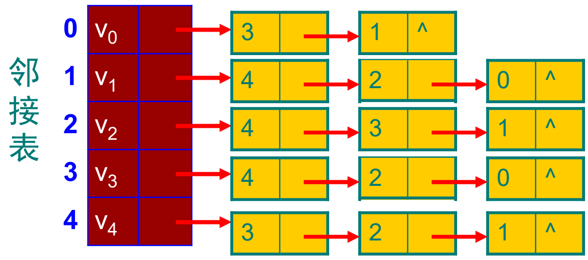
7.4 图的连通性

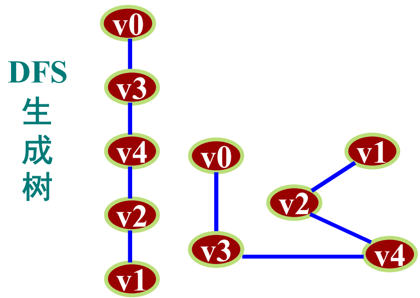
- ❑ 利用图的遍历运算求解图的连通性问题
  - ❖ 无向图是否连通、有几个连通分量，求解无向图的所有连通分量
    - ▪ 深度优先生成树、生成森林
    - ▪ 广度优先生成树、生成森林
  - ❖ 有向图是否是强连通、求解其强连通分量
- ❑ 求无向网的最小代价生成树

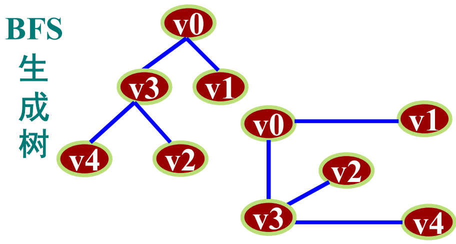
82

例1：画出下图的生成树

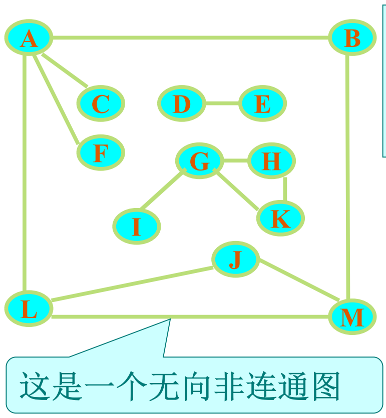








例2：画出下图的生成森林（或极小连通子图）



生成森林（对有向或无向图均适用）：是若干棵生成树的集合，含全部顶点，但构成这些树的边或弧是最少的。

求解步骤：

Step1:先给出邻接矩阵或邻接表；

Step2:写出DFS或BFS结果序列；

Step3:画出对应子图或生成森林。

其实由邻接矩阵或邻接表也能直接画出生成森林

下面选用邻接表方式来求深度优先搜索生成森林

先写出邻接表（或邻接矩阵）：

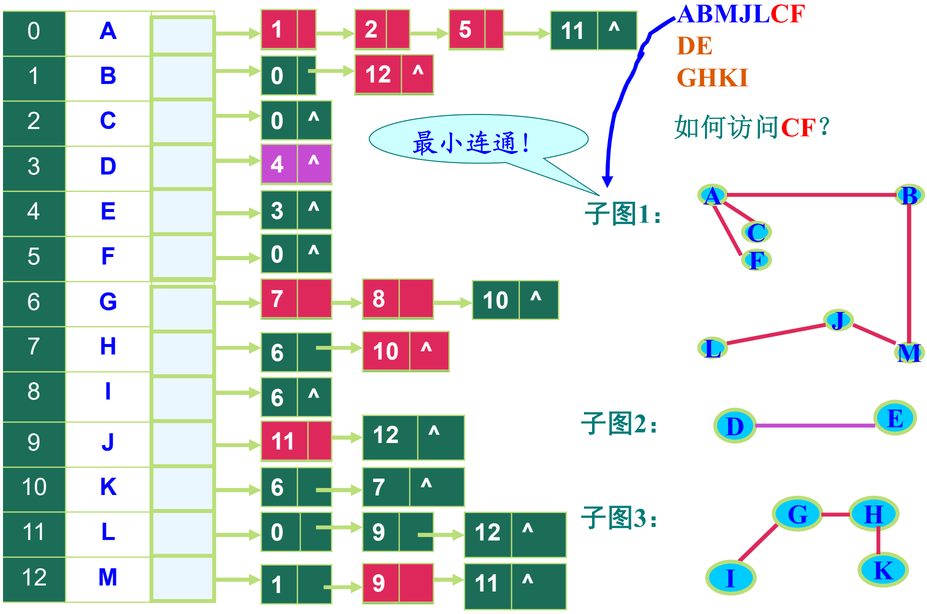
再写出DFS结果

ABMJLCF

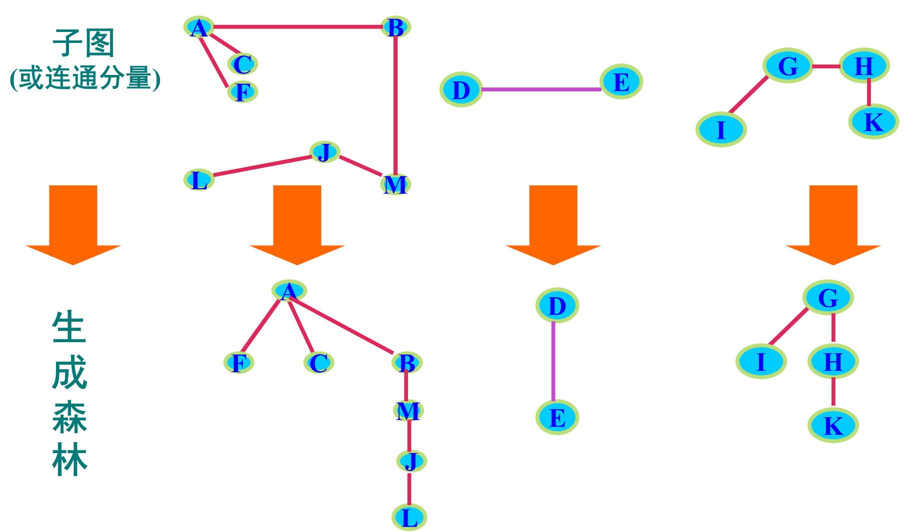
DE

GHKI

如何访问CF?



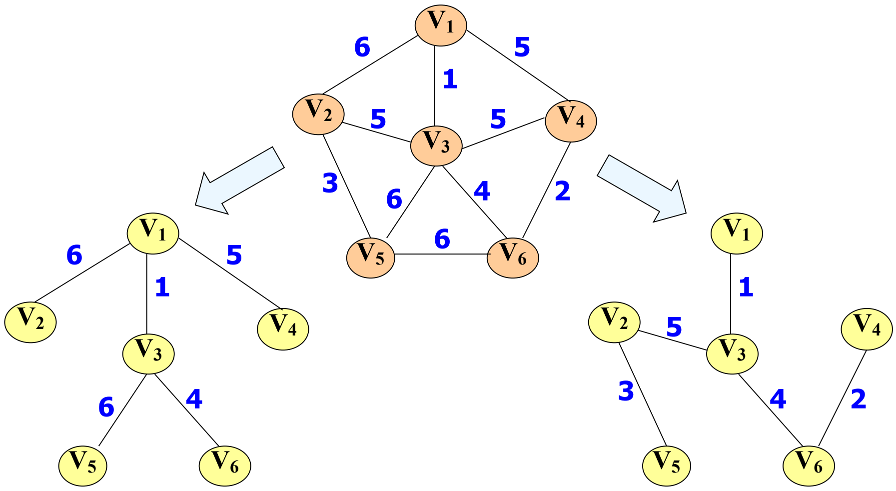
85



86

7.4.3 (连通网的)最小代价生成树

❑ 生成树的代价等于其边上的权值之和。



87

问题:

假设要在 n 个城市之间建立通讯联络网，则连通n 个城市只需要修建 n-1条线路，如何在最节省经费的前提下建立这个通讯网?

该问题等价于:

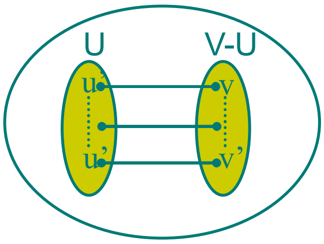
构造网的一棵最小生成树，即:
在 e 条带权的边中选取 n-1 条边（不构成回路），使“权值之和”为最小。

- ❖ Kruskal（克鲁斯卡尔）算法
- ❖ Prim（普里姆）算法
- ❖ Boruvka算法（并行算法）

88

最小生成树性质（Light-Edge Property)

设N=（V，{E}）是一个连通网，
U是顶点集V的一个非空子集。
若（u，v）是一条具有最小权值的边，其中u∈U,v∈V-U,
即（u，v）=Min{cost(x,y)|x∈U,y∈V-U}
则必存在一棵包含边（u，v）的最小生成树。



含义：
将顶点分为两个不相交的互补集合U和V-U，若边cut edge:（u，v）是连接这两顶点集的最小权值边，则边（u，v）必然是某最小生成树的边。

MST Algorithms and Light-Edge Property

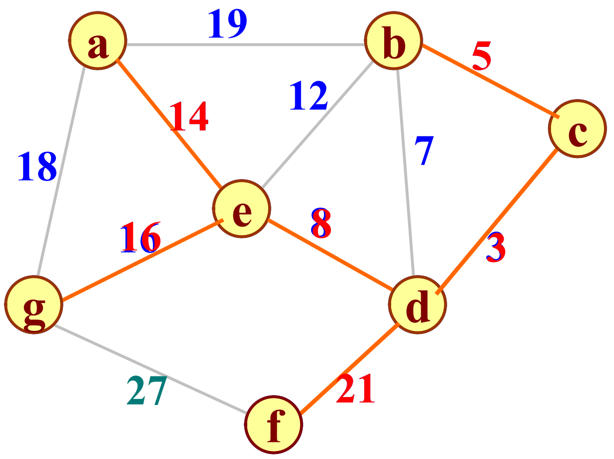
- All three MST greedy algorithms take advantage of the light-edge property.
- Kruskal’s and Prim’s algorithms are based on selecting a single lightest weight edge on each step and are hence sequential, while Boruvka’s selects multiple edges and hence can be parallelized.
- Even though Boruvka’s algorithm is the only parallel algorithm, it was the earliest, invented in 1926, as a method for constructing an efficient electricity network in Moravia in Czech Republic. It was re-invented many times over, the latest one as late as 1965.

普里姆(Prim)算法：归并顶点

- 假设N=(V，E)是连通网，TE是N上最小生成树中边的集合。
- 算法从U={u_0}(u_0∈V)，TE={ }开始，重复执行下述操作：
  - 在所有u∈U，v∈V-U的边(u，v)中找一条代价最小的边(u_0 , v_0)，将其并入集合TE，同时将v_0并入U集合。
  - 当U=V则结束，此时TE中必有n-1条边，则T=(V，{TE})为N的最小生成树。
- Prim算法构造最小生成树的过程是从一个顶点U={u_0}作初态，不断寻找与U中顶点相邻且代价最小的边的另一个顶点，并扩充到U集合直至U=V为止。

91

例如：



所得生成树权值和 = 14+8+3+5+16+21 = 67

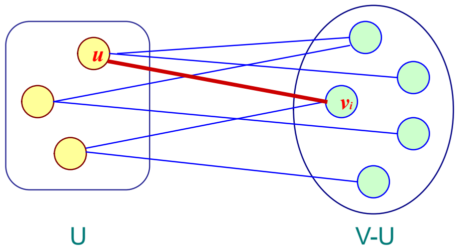
92

一般情况下所添加的顶点应满足下列条件:

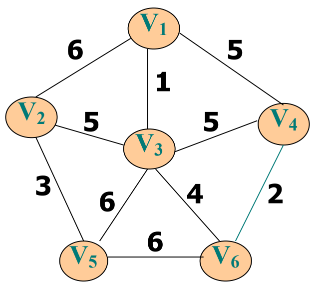
在生成树的构造过程中，图中 n 个顶点分属两个集合：

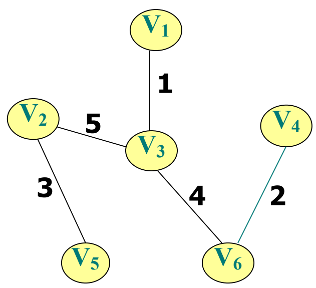
- 已落在生成树上的顶点集 U
- 尚未落在生成树上的顶点集V-U

则应在所有连通U中顶点和V-U中顶点的边中，选取权值最小的边。（基于贪心思想选择顶点）



$$\min_{v_i \in V-U} \min_{u \in U} \text{cost}(u, v_i)$$





步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$
(3)	$\{V_1, V_3, V_6, V_4\}$	$\{V_2, V_5\}$
(4)	$\{V_1, V_3, V_6, V_4, V_2\}$	$\{V_5\}$
(5)	$\{V_1, V_3, V_6, V_4, V_2, V_5\}$	$\{\ \}$

设置一个辅助数组，对当前V—U集中的每个顶点，记录和顶点集U中顶点相连接的代价最小的边：

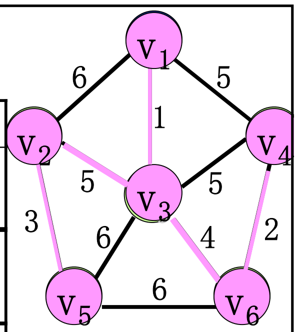
```
struct {
    VertexType  adjvex;          // U集中的顶点
    VRType     lowcost;          // 边的权值
} closedge[MAX_VERTEX_NUM];
```

$\forall\ v_i \in V - U,$

$$\text{closedge[i-1].lowcost} = Min\{\cos t(u, v_i) \mid u \in U\}$$

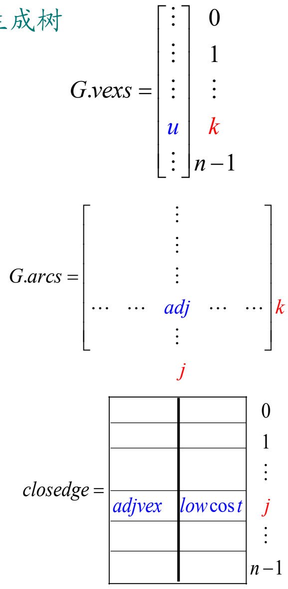
普里姆算法过程示例：

	2	3	4	5	6	U	k
Adjvex lowcost	v1 6	v1 1	v1 5			{v1}	3
Adjvex lowcost	v3 5	0	v1 5	v3 6	v3 4	{v1, v3}	6
Adjvex lowcost	v3 5	0	v6 2	v3 6	0	{v1,v3,v6,}	4
Adjvex lowcost	v3 5	0	0	v3 6	0	{v1,v3,v6, v4}	2
Adjvex lowcost	0	0	0	v2 3	0	{v1,v3,v6, v4,v2}	5
Adjvex lowcost	0	0	0	0	0	{v1,v3,v6, v4,V2,v5}	



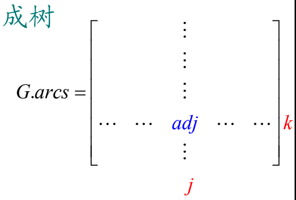
```
void MiniSpanTree_PRIM (MGraph G, VertexType u){
  k=LocateVex(G,u);   //从顶点u出发构造G的最小生成树
  for(j = 0; j < G.vexnum; ++j)  //辅助数组初始化
     if ( j != k ) closedge[j] = {u, G.arcs[k][j].adj};
  closedge[k].lowcost = 0;      //初始, U={u}

}// MiniSpanTree_PRIM
```



97

```
void MiniSpanTree_PRIM (MGraph G, VertexType u){
  k=LocateVex(G,u);   //从顶点u出发构造G的最小生成树
  for(j = 0; j < G.vexnum; ++j)   //辅助数组初始化
     if ( j != k ) closedge[j] = {u, G.arcs[k][j].adj};
  closedge[k].lowcost = 0;         //初始, U={u}
  for(i = 1; i < G.vexnum; ++i) {  //选择其余G.vexnum-1个顶点
     k = minimum(closedge);      //求生成树的下一个顶点k
     printf(closedge[k].adjvex, G.vexs[k]);  //输出生成树的边
     closedge[k].lowcost = 0;     //顶点k并入U集合
     for(j = 0; j < G.vexnum; ++j)
        if (G.arcs[k][j] < closedge[j].lowcost) //顶点k并入U后更新数组
           closedge[j] = {G.vexs[k], G.arcs[k][j].adj};
  }//for
}// MiniSpanTree_PRIM
```



算法的时间复杂度为：$O(n^2)$

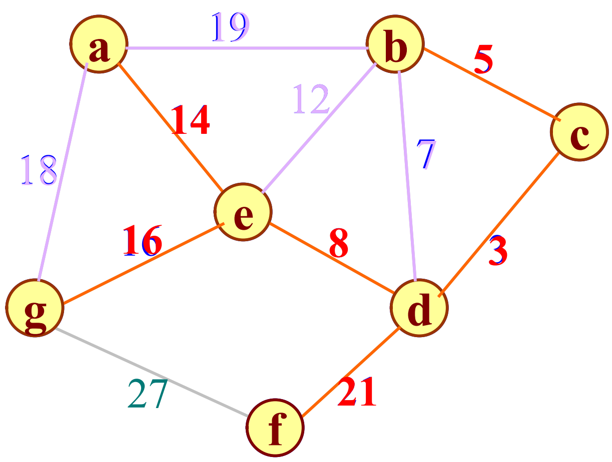
98

克鲁斯卡尔(Kruskal)算法: 归并边

- 假设连通网N=(V, E)，则令最小生成树的初始状态为只有n个顶点而无边的非连通图T=(V, {})，图中每个顶点自成一个连通分量。
- 在E中选择代价最小的边，然后判断：
  - 若该边依附的顶点落在T中不同的连通分量上，则将此边加入到T中，
  - 否则舍去此边而选择下一条代价最小的边。
- 依次类推，直至T中所有顶点都在同一连通分量上为止(此时，T有n-1 条边）。
- 实现时，可先对e条边进行排序。

99

例如:



100

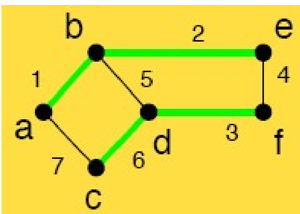
两种串行算法比较

算法名	普里姆算法	克鲁斯卡尔算法
时间复杂度	$O(n^2)$	$O(eloge)$
适应范围	稠密图	稀疏图

101

Parallel Minimum Spanning Tree

- Kruskal and Prim's algorithm are sequential algorithms. Both picked light edges belonging to MST carefully one by one.
- It is possible to select many light edges at the same time.
- By the light edge rule, for each vertex, the minimum weight edge between it and its neighbors is in the MST. These edges are referred as the ***minimum-weight edges*** of the graph.

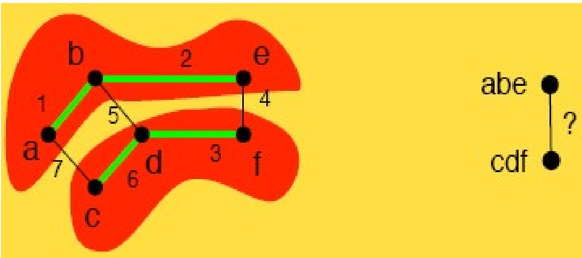


- A way to find some edges in the MST in parallel.
- Can find all the MST edges?
- Given that we have found some of the edges, how can we proceed?

102

Parallel Boruvka Algorithm using Graph Contraction

- **Graph contraction:** Given a partition of the graph into disjoint connected subgraphs, then replace each subgraph (partition) with a supervertex and relabel the edges. This is repeated until no edges remain.
- **The idea of Boruvka's algorithm** is to use graph contraction to collapse each component that is connected by a set of minimum-weight edges into a single vertex.



- How to process redundant edges?
- Here only keep the minimum of the redundant edges.

Boruvka Algorithm using Graph Contraction

Boruvka's algorithm

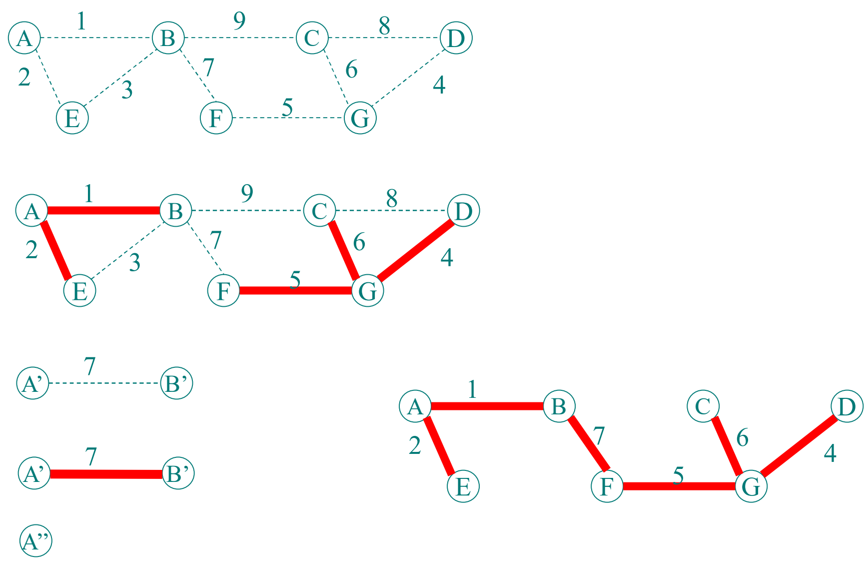
While there are edges remaining:

(1) select the minimum weight edge out of each vertex and contract each connected component defined by these edges into a vertex;

(2) remove self edges (delete loops), and when there are redundant edges keep the minimum weight edge;

(3) add all selected edges to the **MST**.

Boruvka Algorithm Step Illustration



7.5　有向无环图及其应用

一个无环的有向图称为有向无环图（directed acyclic graph），简称DAG图。

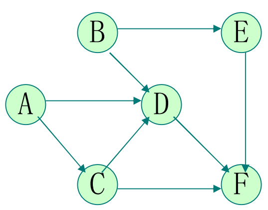


图1

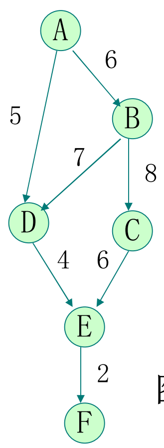


图2

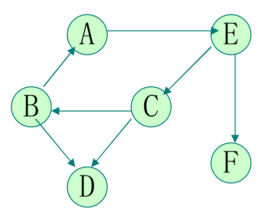


图3（非DAG）

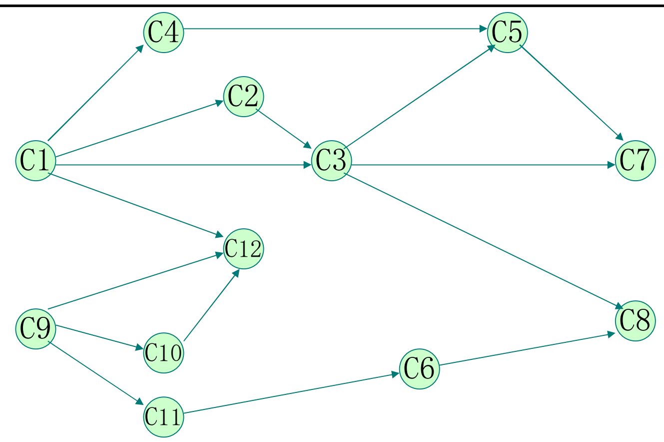
7.5.1 拓扑排序

AOV网（Activity On Vertex network）：

以顶点表示活动，弧表示活动之间的优先关系的DAG图。

计算机软件相关课程

课程编号	课程名称	先决条件
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1，C2
C4	汇编语言	C1
C5	语言的设计和分析	C3，C4
C6	计算机原理	C11
C7	编译原理	C5，C3
C8	操作系统	C3，C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C9，C10，C1



表示课程间关系的有向图

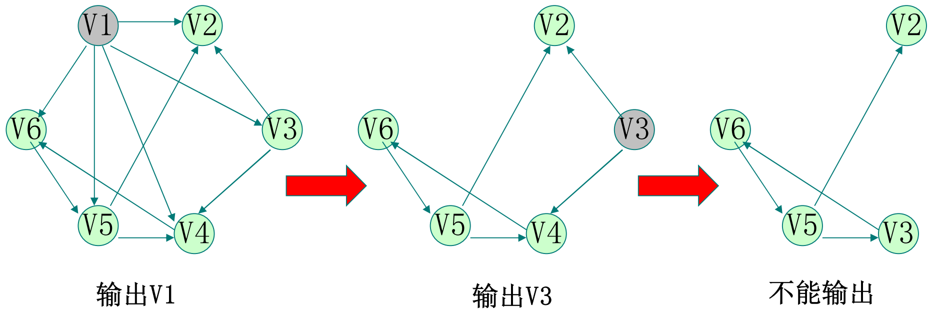
拓扑排序:是有向图的全部顶点的一个线性序列，该序列保持了原有向图中各顶点间的相对次序。例：

（C1, C2, C3, C4, C5, C7, C9, C10, C11, C6, C12, C8）

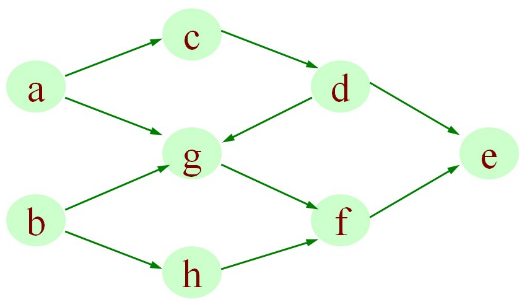
（C9, C10, C11, C6, C1, C12, C4, C2, C3, C5, C7, C8）

若在有向图中有弧<v,u>，则称v是u 的前趋， u是v的后继。

- 拓扑排序方法：
  1) 在有向图中选一无前趋的顶点v(入度为0的顶点)，输出之；
  2) 从有向图中删除v及以v为尾的弧(v的邻接顶点入度减1)；
  3) 重复1)、2)，直至输出全部顶点或有向图中不存在无前趋的顶点时为止。
- 有回路的有向图不存在拓扑排序。



109



a b h c d g f e

拓扑排序涉及的数据和操作：

数据：有向图，顶点的入度；

操作： （1） 选择一入度为 0 顶点v，输出；

（2） 将 v 邻接到的顶点 u 的入度减1；

110

❑ 实现分析

1. 用邻接表存储有向图

头结点中增加记录顶点入度的域；

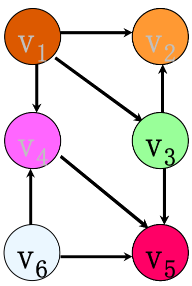
或建立辅助数组存储顶点的入度。

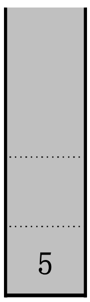
2. 每输出一个入度为**0**的顶点，则将其邻接点的入度减**1**；

3. 为避免重复检测入度为**0**的顶点，如何处理？

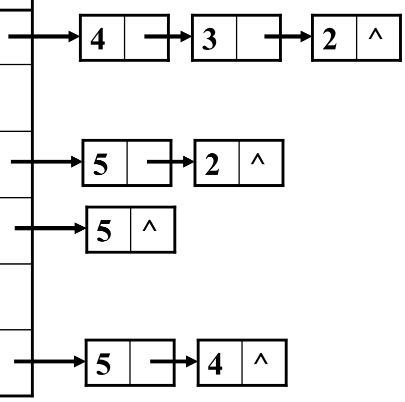
可设一个栈暂存所有入度为**0**的点,

所有入度为**0**的顶点先入栈，出栈后输出。





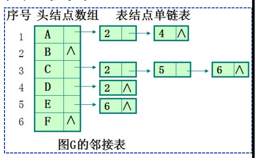
下标	入度	data	first arc
1	0	v_1	
2	0	v_2	^
3	0	v_3	
4	0	v_4	
5	0	v_5	^
6	0	v_6	

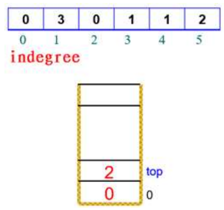


输出：v6 v1 v4 v3 v2 v5

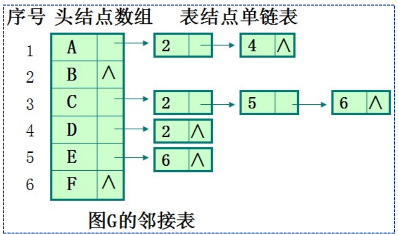
拓扑排序是一种对非线性结构的有向图进行线性化的重要手段

```
Status  TopologicalSort( ALGraph G)
{     int St[MAXV], top= -1;          //栈St的top指针指向栈顶元素
     FindInDegree(G, indegree);   //求图G各顶点的入度
     for (i=0; i<G.vexnum; i++)     //入度为0的顶点保存在栈中
         if (! indegree[i])  {   top++; St[top]=i; }
     count = 0;                  //输出顶点计数
     while (top>-1) {          // 栈不为空时循环
         i=St[top];  top--; printf("%d", i);  ++count;
         for( p=G.vertices[i].firstarc; p; p=p->nextarc){
               k = p->adjvex;         //k为i号顶点的邻接点位序
               if ( !(--indegree[k])) {  top++;  St[top]=k;  }
         } //for
     } //while
     if(count<G.vexnum)  return ERROR; //有回路
     else return OK;
}                          时间复杂度为: O(n+e)
```






```
void FindInDegree( ALGraph G,   int *indegree)
{  int i;    ArcNode *p;
    for(i=0; i<G.vexnum; i++)    indegree[i] = 0;    //初始化
    for(i=0; i<G.vexnum; i++) {  //对图G的每个顶点，遍历对应链表
       p=G.vertices[i].firstarc;
       while(p!  =NULL) {
           indegree[p->adjvex]++;   //该顶点的邻接点入度增1
            p = p->nextarc;
       } //while
    } // for
}
```



114

- ❑ 如何判别**AOV**网只有唯一的拓扑排序序列？
- ❑ 如何实现逆拓扑排序：记录逆拓扑排序序列？

利用一个栈：

将拓扑排序序列顶点依次进栈；

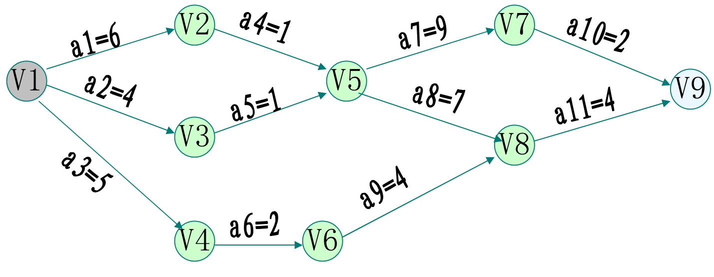
出栈序列就是逆拓扑排序序列。

7.5.2 关键路径

AOE网（Activity On Edge）：

是一个带权的有向无环图，其中以顶点表示事件，弧表示活动，权表示活动持续的时间。

当AOE网用来估算工程的完成时间时，只有一个开始点（入度为0,称为源点）和一个完成点（出度为0，称为汇点）



AOE网研究的问题：

（1）完成整项工程至少需要多少时间；

（2）哪些活动是影响工程进度的关键。

在AOE网中，部分活动可并行进行，所以完成工程的最短时间是从开始点到完成点的最长路径长度。

路径长度最长的路径称为关键路径（Critical Path）。

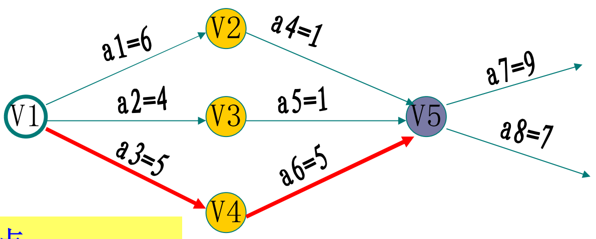
请理解求关键路径的算法及其思想。

117

（顶点）事件vi的最早发生时间ve(i)或ve(vi)：

从开始点到vi的最长路径长度。（ve(v1)=0）

既表示事件vi的最早发生时间，也表示所有以vi为尾的弧所表示的活动ak的最早开始时间e(k)或e(ak)。



仅有一个前驱顶点：

ve(v2)=ve(v1)+6=0+6=6

ve(v3)=ve(v1)+4=0+6=4

ve(v4)=ve(v1)+6=0+5=5

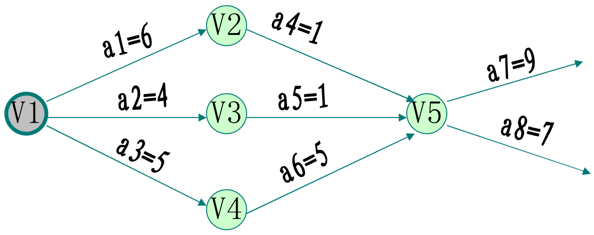
有多个前驱顶点：

ve(v5)=max{ve(前驱顶点)+前驱活动时间}

=max{6+1,4+1,5+5}=10

完成点（汇点）的ve(vn)为工程完成所需要的时间。

118



各顶点事件最早发生时间：

ve(v1)=0

ve(v2)=6

ve(v3)=4

ve(v4)=5

ve(v5)=10

各活动最早开始时间：

e(a1)= e(a2)=e(a3)=ve(v1)=0

e(a4)=ve(v2)=6

e(a5)=ve(v3)=4

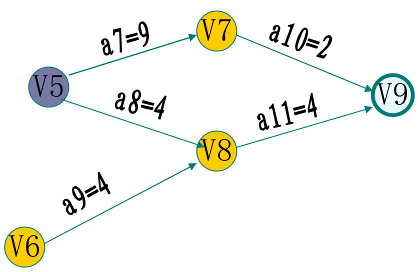
e(a6)=ve(v4)=5

e(a7)=e(a8)=ve(v5)=10

119

不推迟整个工程完成的前提下，（顶点）事件vi允许的最迟发生时间vl(i)或vl(vi)：完成点（汇点）v_n的的最迟发生时间vl(n)减去vi到vn的最长路径长度。

（v_n的最迟发生时间vl(n) = 最早发生时间ve(n)）。



仅有一个后继顶点：

假定工程18天完成(ve(v9)=18)，则：

vl(v9)=18

vl(v7)= vl(v9)-2=16

vl(v8)= vl(v9)-4=14

vl(v6)= vl(v8)-4=10

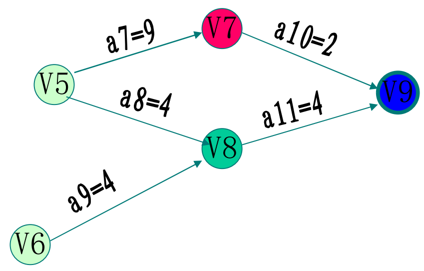
有多个后继顶点：

vl(v5)= min{vl(v7)-9, vl(v8)-4}=min{7,10}=7

120

确定了顶点vi的最迟发生时间后，可确定所有以vi为弧头的活动ak的最迟开始时间l(k)或l(ak)：表示在不推迟整个工程完成的前提下，活动ak最迟必须开始的时间。

l(ak)=vl(ak弧头对应顶点)-活动ak的持续时间



vl(v7)= vl(v9)-2=16

vl(v8)= vl(v9)-4=14

vl(v9)=18

l(a11)=Vl(v9)-4=18-4=14

l(a10)=Vl(v9)-2=18-2=16

l(a9)=Vl(v8)-4=14-4=10

l(a8)=Vl(v8)-4=14-4=10

l(a7)=Vl(v7)-9=-9=716

l(i)-e(i)意味着完成活动ai的时间余量。

关键活动： l(i)=e(i)的活动。

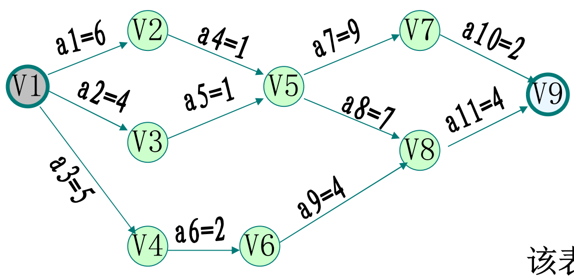
121

关键路径算法步骤：

（1）从开始点v1出发，令ve(1)=0,按拓朴排序序列求其它各顶点的最早发生时间

Ve(k)=max{ve(j)+dut(<j,k>)}

(vj为以顶点vk为弧头的所有弧的弧尾对应的顶点集合)



顶点	ve(i)	vl(i)
v_1	0	
v_2	6	
v_3	4	
v_4	5	
v_5	7，5	
v_6	7	
v_7	16	
v_8	14，11	
v_9	18，18	

该表次序为一拓朴排序序列

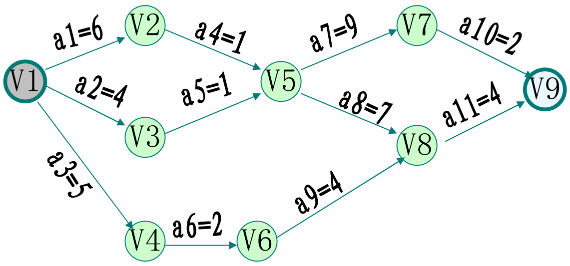
122

关键路径算法步骤：

（2）从完成点v_n出发，令vl(n)=ve(n),按逆拓朴排序序列求其它各顶点的最迟发生时间

Vl(j)=min{vl(k)-dut(<j,k>)}

(vk为以顶点vj为弧尾的所有弧的弧头对应的顶点集合)



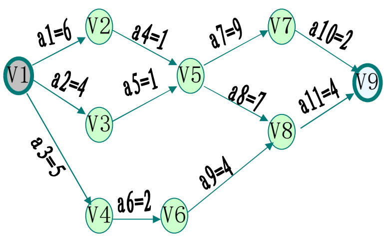
顶点	ve(i)	vl(i)
v_1	0	0，2，3
v_2	6	6
v_3	4	6
v_4	5	8
v_5	7	7，7
v_6	7	10
v_7	16	16
v_8	14	14
v_9	18	18

关键路径算法步骤：

（3）求每一项活动ai(vj,vk)：

e(i)=ve(j)

l(i)=vl(k)-dut(ai)



顶点	ve(i)	vl(i)
v_1	0	0
v_2	6	6
v_3	4	6
v_4	5	8
v_5	7	7
v_6	7	10
v_7	16	16
v_8	14	14
v_9	18	18

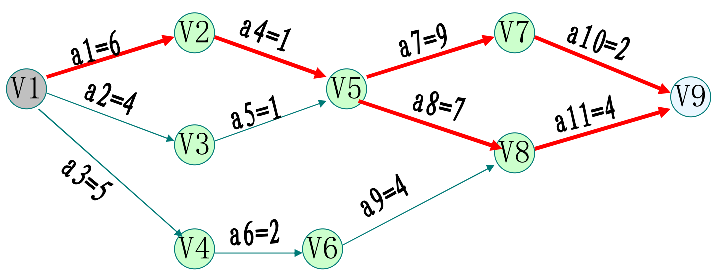
活动	e(i)	l(i)	l(i)-e(i)
a_1	0	0	0
a_2	0	2	2
a_3	0	3	3
a_4	6	6	0
a_5	4	6	2
a_6	5	8	3
a_7	7	7	0
a_8	7	7	0
a_9	7	10	3
a_{10}	16	16	0
a_{11}	14	14	0

关键活动：选取$e(i)=l(i)$的活动。

关键路径：

（1） v1➔ v2 ➔ v5 ➔ v7 ➔ v9

（2） v1➔ v2 ➔ v5 ➔ v8 ➔ v9



求关键路径

1. 输入e条弧<j, k>，建立AOE网的存储结构，顶点数为n，顶点从0开始编号，设源点为v_0、汇点为v_{n-1}
2. 从源点v_0出发，令$v_e[0]=0$，按拓扑有序求其余各顶点事件的最早发生时间$v_e[i]$($1\leqslant i\leqslant n-1$)。如果得到的拓扑有序序列中顶点个数小于网中的顶点数，则说明网中存在环，不能求关键路径，算法终止；否则执行步骤3；
3. 从汇点v_{n-1}出发，令$v_l[n-1]=v_e[n-1]$，按逆拓扑有序求其余各顶点事件的最迟发生时间$v_l[i]$($2\leqslant i\leqslant n-2$)；
4. 根据各顶点的v_e、v_l值，求每条弧s的最早开始时间$e(s)$和最迟开始时间$l(s)$，若某条弧满足条件$e(s)=l(s)$，则为关键活动。

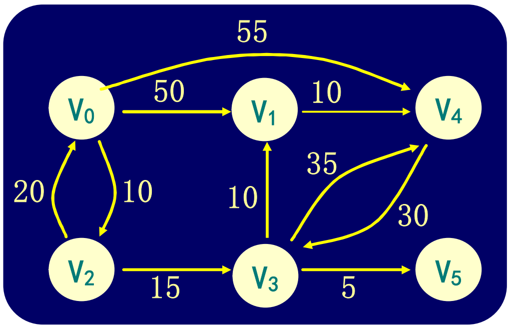
7.6 最短路径

1. 问题的提出

- 交通咨询系统、通讯网、计算机网络需要寻找两结点间最短路径
- 交通咨询系统：A地点到B地点的最短路径
- 计算机网：发送Email节省费用，A到B沿最短路径传送

2. 最短路径及其长度

- 路径长度： 路径上的边数
  路径上边的权值之和
- 最短路径：两结点间权值之和最小的路径



例：求V_0到V_4最短路径

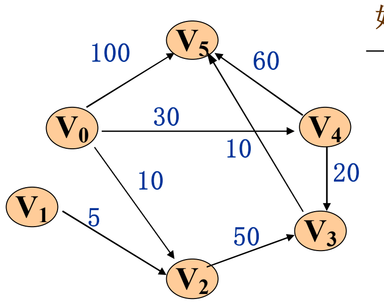
V_0 V_4	55
V_0 V_1 V_4	60
V_0 V_2 V_3 V_4	60
V_0 V_2 V_3 V_1 V_4	45

7.6.1 从某源点到其余各点的最短路径

带权值的有向图的单源最短路径问题.

源点：路径上第一个顶点。

求出从源点到图中其余各个顶点之间的最短路径。



始点	终点	最短路径	路径长度
v_0	v_2	(v_0, v_2)	10
	v_4	(v_0, v_4)	30
	v_3	(v_0, v_4, v_3)	50
	v_5	(v_0, v_4, v_3, v_5)	60

129

迪杰斯特拉算法（Dijkstra）

1.Dijkstra算法的基本思想

- 按路径长度递增顺序求最短路径。

2. Dijkstra 算法的基本步骤

- 设V_0是起始源点，S是已求得最短路径的终点集合。

1) V-S=未确定最短路径的顶点的集合，初始时 S={V_0}，长度最短的路径是边数为1且权值最小的路径。

2) 下一条长度最短的路径：

   ① $V_i \in V - S$，求V_0 到V_i中间只经S中顶点的最短路径；

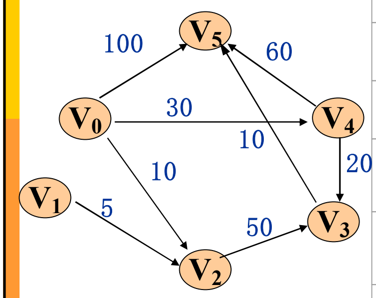
   ② 上述路径中长度最小者即为下一条长度最短的路径；

   ③ 将所求最短路径的终点加入S 中；

3）重复2）直到求出所有终点的最短路径

130

迪杰斯特拉算法（Dijkstra）



- 以V_0为源点

终点	最短路径的求解过程				
	i=1	i=2	i=3	i=4	i=5
v_1	∞	∞	∞	∞	∞
v_2	10 (v_0,v_2)				
v_3	∞	60 (v_0, v_2,v_3)	50 (v_0, v_4,v_3)		
v_4	30 (v_0,v_4)	30 (v_0,v_4)			
v_5	100 (v_0,v_5)	100 (v_0,v_5)	90 (v_0, v_4,v_5)	60 (v_0, v_4,v_3,v_5)	
v_k	v_2	v_4	v_3	v_5	
S	$\{v_0,v_2\}$	$\{v_0, v_2,v_4\}$	$\{v_0, v_2,v_4, v_3\}$	$\{v_0, v_2,v_4, v_3,v_5\}$	

131

迪杰斯特拉算法（Dijkstra）:基于图的邻接矩阵存储

- 设V_0是源点，S是已求得最短路径的终点集合，则V-S是未确定最短路径的顶点的集合；初始时 S =$\{V_0\}$。
- 设D[i]用于保存从源点V_0出发到达顶点V_i 的最短路径长度。
- 初始时，若源点到顶点V_i有边，则D[i]为边上的权值；否则，D[i]为∞。

1) 从V_0出发，长度最短的最短路径是$(V_0\ ,V_k)$，即

$$D[k]=\min\{D[i]\mid V_i\in V-S\}$$

将顶点V_k加入S集合，同时将其从集合V-S中去掉；

132

迪杰斯特拉算法（Dijkstra）:基于图的邻接矩阵存储

2） 求下一条长度最短的路径:

修改从V_0出发到达集合V-S中所有顶点的最短路径的长度，即若

D[k]+arcs[k][i]<D[i]　　($V_i \in$ V-S)，则

D[i]=D[k]+arcs[k][i]

上述最短路径中长度最小者即为下一条长度最短的路径，即

D[k]=min{D[i]|$V_i \in$ V-S}

将顶点V_k加入S集合，同时将其从集合V-S中去掉;

3） 重复2）直到求出所有顶点的最短路径。

Dijkstra算法总的时间复杂度为O(n^2)

133

7.6.2 每一对顶点之间的最短路径

算法1（Dijkstra算法）：　总的时间复杂度为**O(n^3)**

以每一个顶点为源点，重复执行Dijkstra算法n次，即可求出每一对顶点之间的最短路径。

算法2（Floyd算法）：　总的时间复杂度为**O(n^3)**

算法思想：（以邻接矩阵作为图的存储结构）

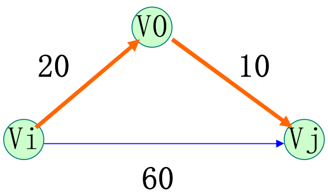
假设求V_i到V_j的最短路径，如果从V_i到V_j有弧，则存在一条长度为arcs[i][j]的路径，该路径不一定是最短路径，尚需进行n次试探。

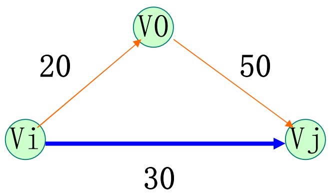
134

❑首先考虑(Vi,V0,Vj)是否存在

✓即判断(Vi,V0)和(V0,Vj)是否存在，

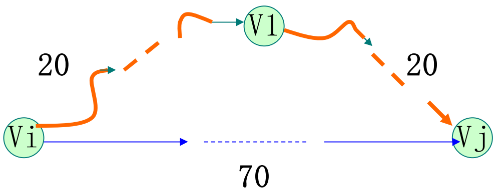
✓如果存在，比较(Vi, Vj)和(Vi,V0)+(V0,Vj)，取长度较短的为从Vi到Vj的中间顶点序号不大于0的最短路径。





再考虑路径上再增加一个顶点V1，如果考虑(Vi,…V1)和(V1,…. Vj)，(Vi,…V1)和(V1,…. Vj)都是中间顶点序号不大于0的最短路径。(Vi,…V1，…. Vj)可能是从Vi到Vj的中间顶点序号不大于1的最短路径。

比较Vi到Vj的中间顶点序号不大于0的最短路径和(Vi,…V1)+(V1,…. Vj)，取长度较短的为从Vi到Vj的中间顶点序号不大于1的最短路径。



以此类推，经过n次比较后，求得Vi到Vj的最短路径。

假定图G顶点之间的最短路径长度矩阵为D[n][n]，初始化为其邻接矩阵G.arcs.

Floyd算法的基本思想是递推产生一个方阵序列：

$D^{(-1)}, D^{(0)}, D^{(1)}, \ldots\ldots, D^{(k)}, \ldots. D^{(n-1)}$

$D^{(-1)}[i][j] = D[i][j] = G.arcs[i][j]$

$D^{(k)}[i][j] = \mathrm{Min}\{D^{(k-1)}[i][j],\ D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\}$

$0 \leqslant i, j, k \leqslant n-1$

算法C程序：

```
for(k=0;k<n;k++)          //依次选定中间顶点V0,V1,...Vn-1
 for(i=0; i<n;i++)        //i,j配合处理所有顶点Vi,Vj
   for(j=0; j<n;j++)
     if (D[i][j]>D[i][k]+D[k][j])
         D[i][j]=D[i][k]+D[k][j]; //取较短路径
```


Dijkstra算法与Floyd算法分析

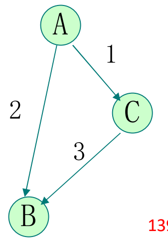
- **Dijkstra**算法是串行算法
- **Floyd**算法能否并行化？采用一种结构表示图，思考并行**Floyd**算法的实现方法。

讨论问题

1. 带权图（权值非负，表示边连接的顶点间的距离）的最短路径问题是找出从初始顶点到目的顶点之间的一条最短路径，假设从初始顶点到目标顶点之间存在路径，现有解决问题的方法：

（**1**）该最短路径初始时仅包含初始顶点，令当前顶点**u**为初始顶点；（**2**）选择离**u**最近且尚未在最短路径中的一个顶点**v**，加入到最短路径中，修改当前顶点**u=v**；（**3**）重复步骤（**2**），直到**u**是目标顶点为止。

请问上述方法能否求得最短路径？若该方法可行，请证明之；否则请举例说明。（**10**分）



41. （8 分）已知有一个 6 个顶点（顶点编号 0～5）的有向带权图 G，其邻接矩阵 A 为上三角矩阵，它的压缩存储如下：

4	6	∞	∞	∞	5	∞	∞	∞	4	3	∞	∞	3	3

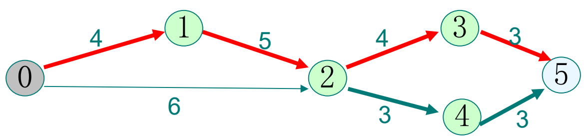
要求：

(1) 写出图 G 的邻接矩阵 A；

(2) 画出有向带权图 G；

(3) 求图 G 的关键路径，并计算关键路径的长度。

$$
\begin{pmatrix}
\infty & 4 & 6 & \infty & \infty & \infty \\
 & \infty & 5 & \infty & \infty & \infty \\
 & & \infty & 4 & 3 & \infty \\
 & & & \infty & \infty & 3 \\
 & & & & \infty & 3 \\
 & & & & & \infty
\end{pmatrix}
$$



41. [13分]已知优先图G采用邻接矩阵存储是，其定义如下

```
Typedef struct{
Int numberVertices,numEgges;
Char VerticesList[maxV];
Int edge[maxV][maxV];
}MGraph;
```

将图中出度大于入度的顶点成为K顶点，如图，a和b都是k顶点

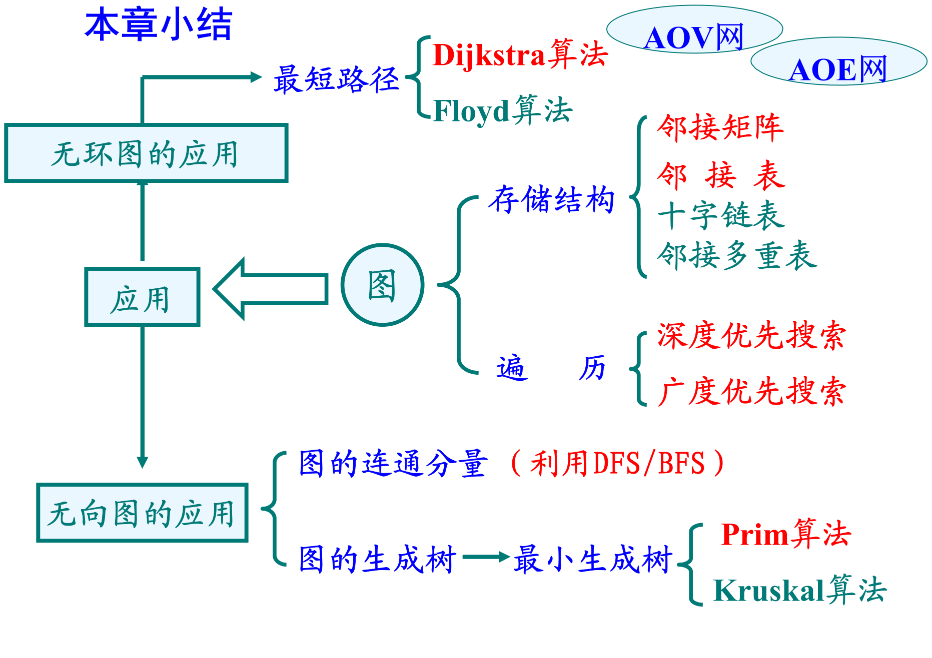
设计算法int printVertices(MGraph G)对给定任意非空有向图G,输出G中所有K顶点的算法，并返回K顶点的个数。

(1)给出算法的设计思想。

(2)根据算法思想，写出C/C++描述，并注释。

141

本章小结



142

课外思考

严：7.1　　7.7　　7.10

（ 7.5　7.22 ）

143