第5章 图

- **主要内容：**
- 图的基本概念
- 图的存储结构
- 图的遍历

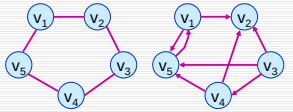
图的概述

- 图是一种比树形结构更复杂的非线性结构。
  - 在线性结构中，每个数据元素至多有一个直接前驱和一个直接后继。
  - 在树型结构中，每个数据元素至多有一个直接前驱，但可以有多个直接后继。
  - 在图结构中，每个结点可以和其它任何结点相关联。

线性结构：数据元素之间仅具有线性关系(前驱和后继)
树结构：结点之间具有层次关系(双亲和孩子)
图结构：任意两个结点之间都可能有关系(邻接)

§5.1 图的定义和基本概念

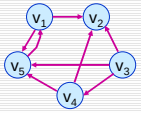
- 1. 图的定义
- **图(Graph)：** 图G由两个集合V(G)和E(G)组成，记为Graph=(V,E)
  - V(G)是顶点的非空有限集
  - E(G)是边的有限集合，边可以是顶点的无序对或有序对



E(G)可以是空集。若E(G)为空，则图G只有顶点而没有边

有向图

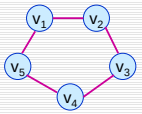
- **有向边**：若从顶点v_i到v_j的边有方向，则称这条边为有向边(也称弧)，表示为$<v_i,v_j>$。其中v_i为弧尾(初始点)，v_j为弧头(终端点)，在图示时用箭头指明方向。
- **有向图**：若图G的任意两个顶点之间的边都是有向边，则称G为有向图。
- 例如：
  - $V(G)=\{v_1, v_2, v_3, v_4, v_5\}$
  - $E(G)=\{<v_1,v_2>,<v_3,v_2>,<v_3,v_4>,<v_4,v_5>,<v_1,v_5>,<v_5,v_1>,<v_3,v_5>,<v_4,v_2>\}$



注意：$<v_i,v_j>$和$<v_j,v_i>$是两条不同的有向边

无向图

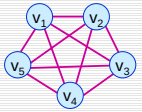
- **无向边**：若从顶点v_i到v_j的边没有方向，则称这条边为无向边，表示为(v_i,v_j)。
- **无向图**：若图G的任意两个顶点之间的边都是无向边，则称G为无向图。
- 例如：
  - $V(G)=\{v_1, v_2, v_3, v_4, v_5\}$
  - $E(G)=\{(v_1,v_2), (v_2,v_3), (v_3,v_4), (v_4,v_5), (v_1,v_5)\}$

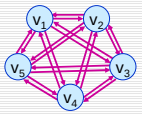


注意：
- (v_i,v_j)和(v_j,v_i)是同一条边
- 边$(v_i,v_j)\in E(G)$ 必有$(v_j,v_i)\in E(G)$, 即E(G)是对称的

完全图

- **无向完全图**：如果在无向图G中，任何两顶点都有一条边相连，则称为无向完全图或**完全图**。
  - 完全图恰有$n(n-1)/2$条边
- **有向完全图**：如果在有向图G中，任何两顶点都有两条方向相反的弧线连接，则称为有向完全图
  - 完全有向图恰有$n(n-1)$条边
- **稠密图**：当一个图接近完全图时，则称它为稠密图
- **稀疏图**：当一个图中含有较少的边或弧时，则称它为稀疏图。





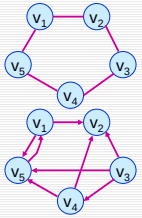
顶点和边数的关系

- 若G是一般无向图，则边数

$$0 \le e \le \frac{n(n-1)}{2}$$

- 若G是一般有向图，则边数

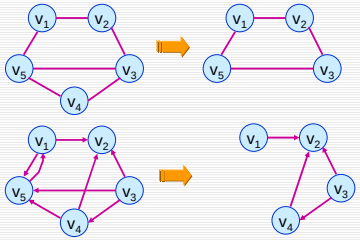
$$0 \le e \le n(n-1)$$



注意：在图的讨论中做了一些限制
- 第一，图中不能有从顶点自身到自身的边(即自身环)，就是说不应有形如(v_x,v_x)的边或$<v_x,v_x>$的弧。
- 第二，两个顶点v_t和v_u之相关联的边不能多于一条。

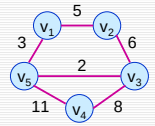
子图

- **子图**：设有两个图，G=(V,E)，G'=(V',E')；若V'∈V，且E'∈E，则称G'为G的子图。



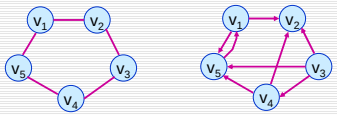
2. 图的基本术语

- **权**：反映图的边或顶点的某些特性的数据叫作权。
- **网**：若G中的每一条边都有反映这条边的某种特性的权值，则称该图为网。权值所反映的特性，由具体问题决定，可以是距离，时间，价格等。



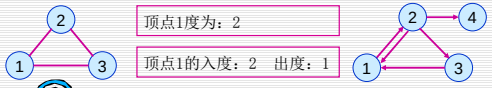
图的基本术语(.续)

- **邻接点**
  - 在无向图中，若边$(v_i,v_j)\in E(G)$，则称顶点v_i和v_j互为邻接点，边(v_i,v_j)**依附**于顶点v_i和v_j，或者说边(v_i,v_j)和顶点v_i和v_j**相关联**
  - 在有向图中，若弧$<v_i,v_j>\in E(G)$，则称顶点v_i**邻接到**顶点v_j，顶点v_j**邻接自**顶点v_i。



图的基本术语(..续)

- **顶点的度**：记为TD(v)
  - 无向图中顶点的度，是依附于该顶点的边的数目
  - 有向图中顶点的度，为入度与出度之和。
    - 入度：以该顶点为终点的弧的数目，记为ID(v)
    - 出度：以该顶点为始点的弧的数目，记为OD(v)



顶点1度为：2

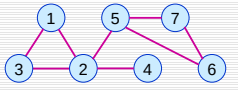
顶点1的入度：2　出度：1

顶点的度TD(v_i)和边数e有什么关系？

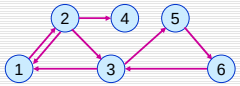
$$e=\frac{1}{2}\sum_{i=1}^{n}TD(v_i) \qquad e=\sum_{i=1}^{n}ID(v_i)=\sum_{i=1}^{n}OD(v_i)$$

图的基本术语(...续)

- **路径**：路径是图中从顶点v到顶点v'的顶点序列$(v=v_{i,0},\ v_{i,1},\ \ldots\ldots,\ v_{i,m}=v')$，$1\leqslant j\leqslant m$，一般情况下图中路径不唯一
  - 无向图中$(v_{i,j-1},v_{i,j})\in E$
  - 有向图中路径也是有向的，即$<v_{i,j-1},v_{i,j}>\in E$
- **路径长度**
  - 不带权的图：路径上所含边的数目称为路径长度
  - 带权图(网)：路径长度指路径上各边的权之和



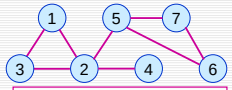
v_1到v_6路径：1, 3, 2, 5, 7, 6
路径长度：5



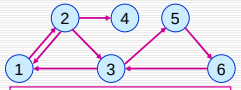
v_1到v_6路径：1, 2, 3, 5, 6
路径长度：4

图的基本术语(....续)

- **简单路径**：顶点序列中，顶点不重复出现的路径称为简单路径
- **回路**：第一个顶点和最后一个顶点相同的路径称为回路或环
- **简单回路**：除第一个和最后一个顶点外，其它顶点不重复出现的回路称为简单回路



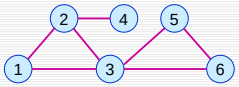
v_1到v_6简单路径：1, 2, 5, 7, 6
回路：1, 2, 5, 7, 6, 5, 2, 1
简单回路：1, 2, 3, 1



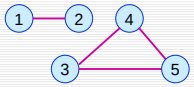
v_1到v_6简单路径：1, 2, 3, 5, 6
回路：1, 2, 3, 5, 6, 3, 1
简单回路：3, 5, 6, 3

3. 图的连通性

- **连通**：在无向图中，若从顶点v_i到顶点v_j有路径，则称v_i和v_j是连通的。
- **连通图**：在无向图中，若任意两点v_i，v_j都有路径相连，称该无向图为连通图。
- **连通分量**：无向图中极大连通子图(包括子图中的所有顶点和所有边)。



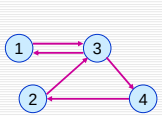
连通图



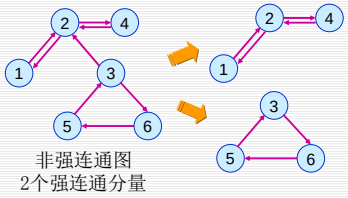
非连通图
2个连通分量

图的连通性(.续)

- **强连通图**：有向图中，若每一对顶点$<v_i, v_j>$，从v_i到v_j，和从v_j到v_i都有路径相连，则称该图为强连通图。
- **强连通分量**：有向图中的极大强连通子图。



强连通图



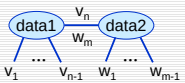
非强连通图
2个强连通分量

抽象数据类型定义

```
ADT  Graph{
  数据对象: V
  数据关系: R
  基本操作:
      Create_G(&G,V,E)：创建图；
      Destroy_G(&G)：销毁图；
      Locate_V(G,v)：返回顶点v在图中的位置；
      Get_V(G,i)：返回第i 个顶点信息；
      First_Adj(G,v)：返回v的第一个邻接点；
      Next_Adj(G,v,w)：返回v的邻接点w的下一个邻接点；
      Insert_V (&G,v)：在图中插入一个顶点v；
      Delete_V (&G,v)：在图中删除一个顶点v；
      Insert_Edge (&G,v,w)：在图中插入一条边<v,w>；
      Delete_Edge (&G,v,w)：在图中删除一条边<v,w>；
      Traverse_G(G)：遍历图中所有顶点；
}ADT  Graph
```


§5.2 图的存储结构

如何表示图中顶点之间的逻辑关系？



图中顶点关系m:n

对于这种任意的逻辑关系，既无法以元素在存储区的物理位置来表示，也不适合在结点中设置多个指针域来实现，这会造成操作过于复杂。因此单纯形式的顺序存储很难实现，而单纯的链式存储也不适合

图的信息构成：顶点、边或弧

分别考虑如何存储顶点、如何存储边

1. 邻接矩阵/数组表示法

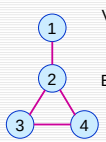
- 邻接矩阵存储法
  - 用一个顺序表来存储顶点信息—顶点表
  - 用邻接矩阵存储顶点间的相邻关系

注意：在简单应用中，可直接用二维数组作为图的邻接矩阵，顶点表可省略

- **图的邻接矩阵**：设图G=(V,{E})，含有n(n≥1)个顶点V={v_1,v_2, ...,v_n}，则边或弧为n×n阶矩阵A(*i*, *j*)，称为图G的邻接矩阵，其中：

$$A(i,j)=\begin{cases}1, & \text{若} <v_i,v_j> \text{或} <v_j,v_i> \in VR \quad i \neq j \\ 0, & \text{反之}\end{cases}$$

邻接矩阵举例



V(G) 1 2 3 4

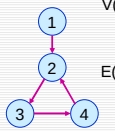
E(G)

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

邻接矩阵特点？

无向图：主对角线为 0 的对称矩阵
有向图：有向完全图对称

如何求顶点V_i的度？



V(G) 1 2 3 4

E(G)

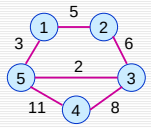
$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

无向图：第i行(或第i列)非零元素的个数
有向图：出度：第i行
入度：第i列

网的邻接矩阵

- 若G＝(V，{E})为网，则邻接矩阵可以表示为：

$$A(i,j)=\begin{cases} w_{i,j}, & <v_i,v_j>或<v_j,v_i>\in VR \quad i\neq j \\ \infty, & 反之 \end{cases}$$



V(G) 1 2 3 4 5

E(G)

$$\begin{bmatrix} \infty & 5 & \infty & \infty & 3 \\ 5 & \infty & 6 & \infty & \infty \\ \infty & 6 & \infty & 8 & 2 \\ \infty & \infty & 8 & \infty & 11 \\ 3 & \infty & 2 & 11 & \infty \end{bmatrix}$$

存储结构类型定义

```
#define INFINITY INT_MAX    //最大值
#define MAX_VERTEX_NUM    20   //最大顶点个数
typedef enum{ DG, DN, AG, AN}GraphKind;
//有向图、有向网、无向图、无向网

typedef struct{
   VertexType vexs[MAX_VERTEX_NUM];  //顶点数组
   Arctype  arcs[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
   //边矩阵，取0, 1或权值w_ij
   int vexnum,arcnum;   //图的当前顶点数和弧数
   GraphKind kind;       //图的种类标志
}MGraph;
```

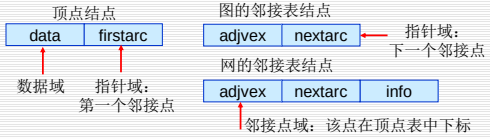

基本操作

- 创建无向图

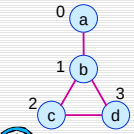
```
status  CreateUDN(MGraph &G )
{  scanf(&G.vexnum,&G.arcnum);
   for (i=0; i<G.vexnum; ++i)  scanf(&G.vexs[i]); //输入顶点
   for (i=0; i<G.vexnum; ++i)
      for (j=0; j<G.vexnum; ++j)
         G.arcs[i][j]=0; //初始化邻接阵
   for (k=0; k<G.arcnum; ++k)
   {   scanf(&v1,&v2,&w);  //输入一条边依附的顶点及权值
      i=locatevex(G, v1);   j=locatevex(G, v2);
      G.arcs[i][j]=w;   G.arcs[j][i]=w;   }
   G.kind=UDN;
   return OK;     }
```

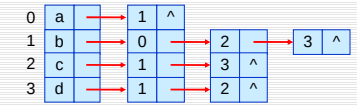

2. 邻接表

- 邻接表表示法
  - 将所有邻接于v_i的顶点v_j链成一个单链表，这个单链表就称为顶点v_i的**邻接表**(无向图也称**边表**，有向图也称**出边表**)
  - 将所有顶点的邻接表头指针和顶点信息放到一维数组中，构成**顶点表**
- 邻接表的两种结点结构



邻接表举例



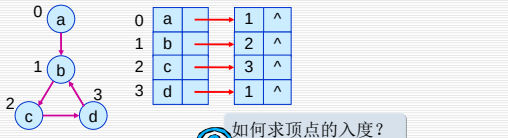


讨论：
边表中的结点表示什么？
如何求顶点的度？
一个图的邻接表是否唯一？

每个结点对应图中的一条边
顶点i的边表中结点的个数
理论上不唯一，计算机实现唯一

总结：无向图中，n个顶点，e条边对应n个表头结点，2e个邻接点

邻接表举例(.续)



如何求顶点的入度？

如何求顶点的出度？

统计各顶点出边表中以顶点*i*为终点的结点个数。O(n×e)

顶点*i*的出边表中结点的个数

或者建立逆邻接链表—以表头结点*i*为终点的单链表，也称入边表

总结：有向图中邻接点数等于边数，即有n个顶点，e条邻接点

存储结构类型定义

```
#define MAX_VERTEX_NUM 20 //最大顶点数
typedef  struct  ArcNode //邻接点
{  int  adjvex;                    //顶点的位置
   struct  ArcNode  *nextarc;
}ArcNode;
typedef  struct  VNode  //顶点
{  vextype  data;
   Arcnode  *firstarc;
}VNode,AdjList[MAX_VERTEX_NUM];
typedef  struct
{  AdjList  vexs;
   int vexnum,arcnum;
   int  kind;            //图的种类标志
}ALGraph
```


邻接矩阵和邻接表总结

- 时间效率
  - 邻接矩阵：建表复杂度为$O(n^2)$。易判断顶点是否相连，易计算出度入度，易读取边信息。
  - 邻接表：建表复杂度$O(n+e)$。易计算出度，不易计算入度，判断顶点是否相连以及获取边信息需要搜索链表
- 空间效率
  - 邻接矩阵
    - 无向图的邻接矩阵是对称矩阵，可压缩存储，需要$n\times(n-1)/2$个存储单元。
    - 有向图的邻接矩阵不一定对称：$O(n^2)$
  - 邻接表：$O(n+e)$
- 唯一性：邻接矩阵唯一；邻接表不唯一
- 适应范围：邻接矩阵-稠密阵；邻接表-稀疏阵

§5.3 图的遍历

- **图的遍历**：指从图中的任一顶点出发，对图中的所有顶点访问一次且只访问一次。

遍历是求解连通性、拓扑排序和关键路径等算法的基础

- 图的遍历操作要解决的关键问题
  - 没有一个"自然"的首结点，任意一个顶点都可第一个被访问

    指定访问的第一个顶点
  - 如果有回路存在，那么一个顶点被访问后，有可能又回到该顶点

    附设访问标志数组visited[0..n-1]
  - 任意一个顶点都可能和其它多个顶点相邻或不相邻，访问这样的结点，还需考虑如何选取下一个出发点

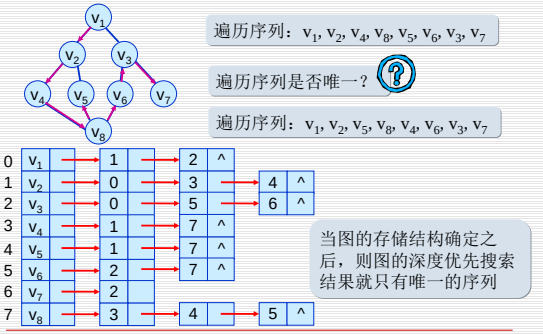
    深度优先遍历和广度优先遍历

1. 深度优先搜索

- 深度优先搜索算法思想：
  - 访问指定的起始顶点v，将其标记为访问过。
  - 从v出发，搜索任意一个和v邻接的未访问过的顶点w，从w出发进行深度优先搜索遍历
  - 重复上两步，直至图中所有和v有路径相通的顶点都被访问到。
  - 若此时图中尚有顶点未被访问，则选图中另一个没有被访问的顶点作为起始点，重复上述过程，直到图中所有顶点都被访问到。

深度优先遍历类似于树的先根遍历，是树的先根遍历的推广

深度优先搜索举例



遍历序列：$v_1, v_2, v_4, v_8, v_5, v_6, v_3, v_7$

遍历序列是否唯一？

遍历序列：$v_1, v_2, v_5, v_8, v_4, v_6, v_3, v_7$

当图的存储结构确定之后，则图的深度优先搜索结果就只有唯一的序列

算法程序

```
int visited[MAX_VERTEX_NUN];

void DFStraverse(Graph  G)
{   for (v=0; v<G.vexnum; ++v) visited[v]=FALSE;
    for (v=0; v<G.vexnum; ++v)    //保证非连通图的遍历
        if (!visited[v]) DFS(G,v);   }

void DFS(Graph G, int v)          //连通图的遍历
{   visit(v);  visited[v]=TRUE;
    for (w=First_AdjVex(G,v); w; w=Next_AdjVex(G,v,w))
        if (!visited[w])  DFS(G,w);   }
```


算法复杂度分析

- 在遍历时，对图中每个顶点至多调用一次DFS 函数，因此，遍历图的过程实质上是对每个顶点查找其邻接点的过程，或者说是通过边或弧找邻接点的过程。
- 算法时间复杂度：与图的存储结构有关
  - 邻接矩阵：查找一个顶点的邻接点所需时间为$O(n)$，查找所有顶点的邻接点所需时间为$O(n^2)$。因此总的复杂度为$O(n^2)$。
  - 邻接表：查找邻接点的时间复杂度为$O(e)$，e为无向图中的边数或有向图中的弧数。所以整个算法的时间复杂度为$O(n+e)$。

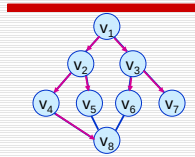
2. 广度优先搜索

- 广度优先搜索算法思想：
  - 访问指定的起始顶点v
  - 依次访问v的各个未曾访问的顶点w_1, w_2, …, w_t
  - 按w_1, w_2, …, w_t的顺序，依次访问其中每一个顶点的所有未访问过的邻接点，并使“先被访问顶点的邻接点”先于“后被访问顶点的邻接点”被访问，直至图中所有与顶点v有路径相通的顶点都被访问到。
  - 若此时图中尚有顶点未被访问，则另选一个未被访问的顶点为起点，重复上述步骤，直到图中所有顶点都被访问到。

广度优先遍历类似于树的按层次遍历，是层次遍历的推广

层次遍历需设置一个队列，每访问一个顶点，就把它的未访问过的邻接点送入队列中。重复上述步骤，直到队列为空。

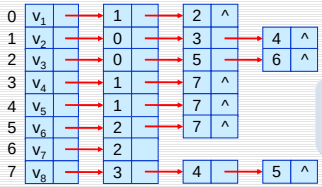
广度优先搜索举例



遍历序列：$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$

遍历序列是否唯一？

遍历序列：$v_1, v_2, v_3, v_5, v_4, v_6, v_7, v_8$



当图的存储结构确定之后，则图的广度优先搜索结果就只有唯一的序列

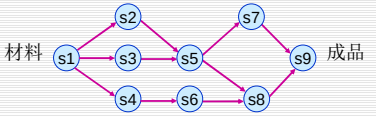
算法程序

```
void BFStraverse(Graph  G)
{  for(v=0; v<G.vexnum; ++v) visited[v]=FALSE;
   InitQueue(Q);
   for(v=0; v<G.vexnum; ++v)  //保证非连通图的遍历
     if(!visited[v])
     {  visit(v);  visited[v]=TRUE;  Enqueue(Q,v);
        while(!Queueempty(Q))
        {  Dequeue(Q,u);
           for(w=First_AdjVex(G,u); w; w=Next_AdjVex(G,u,w))
              if(!visited[w])
              {   visit(w);  visited[w]=TRUE; Enqueue(Q,w);   }
     }
}
```

算法时间复杂度与深度优先相同，空间复杂度$O(n)$

§5.4 有向无环图及其应用

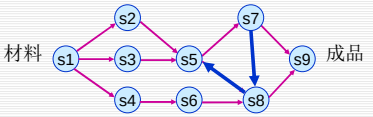
- **有向无环图**：无环的有向图，简称DAG图。
- 描述一项工程或系统的进行过程
  - 几乎所有的工程(Project)都可分成若干个称为活动(activity)的子工程，这些子工程之间通常受到一定的条件约束，如某些子工程的开始必须在另一些子工程完成之后



- 对整个工程和系统，关键是两个方面的问题：
  - 工程能否顺利进行—拓扑排序
  - 估算整个工程完成所必须的最短时间—关键路径

AOV网与拓扑排序

- **AOV网**：以顶点表示活动，弧表示活动间的优先关系的有向图，称为以顶点表示活动的网(Activity On Vextex network)，即AOV网



出现回路意味着什么？

意味着某项活动以自己为先决条件

AOV网的特点：
- 弧表示活动之间存在的某种制约关系。
- 不允许有回路。

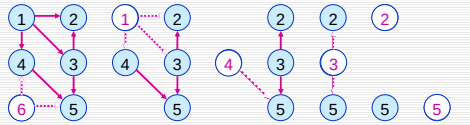
拓扑排序

- **拓扑有序序列**：对于一个有向图，如果存在一个包含其所有顶点的线性序列，该序列不仅保持图中各顶点间原有的先后顺序，而且任意两个顶点间有且仅有单一的先后顺序，这样的线性序列称为拓扑有序序列。
- **拓扑排序**：按照有向图各顶点间相互之间的优先关系，构造拓扑有序序列的操作。

某个有向图，如果能构造它的拓扑序列，则该图中不存在有向回路。因为若图中存在有向环，无论如何安排，必定至少有一条边与其余边是反向的，该图的拓扑序列不存在。

拓扑排序方法

- 步骤
  - 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

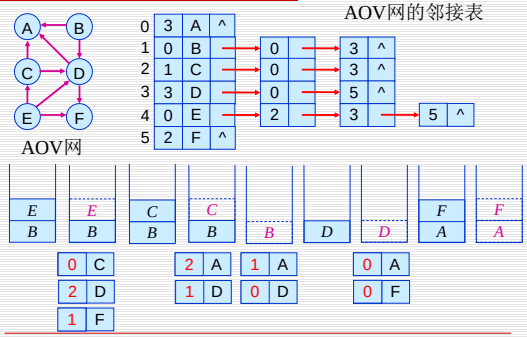


一个AOV网的拓扑有序序列不是唯一的

算法描述

- 以邻接表作存储结构
- 为便于算法执行过程中考察每个顶点的入度，即查找没有前驱的顶点，头结点增加一个数据域存放顶点入度，或者用一个局部向量indegree[0...n-1]来保存各顶点当前的入度
- 为了避免每一次选无前驱顶点时出现重复扫描，设置一个栈(或队列)来存储所有入度为0的顶点，即把邻接表中所有入度为0的顶点依次进栈
- 栈非空时，输出栈顶元素v_j并退栈；在邻接表中查找v_j的直接后继v_k，把v_k的入度减1；若v_k的入度为0则进栈，重复上述操作直至栈空为止。
- 若栈空时输出的顶点个数不是n，则有向图有环；否则，拓扑排序完毕

举例



§5.5 最短路径

- **最短路径**：带权图中，指两顶点间所经过的边上的权值之和为最小的路径。非网图中，指路径上经过的边的数目最小的路径。
- 应用：假设用顶点表示城市，边表示城市间的公路，则由这些顶点和边组成的图可以表示沟通各城市的公路网。若把两个城市之间的距离或该段公路的养路费等作为权值，赋于图中的边，就构成了一个带权的图。汽车司机关心的问题是：
  - 从甲地到乙地是否有公路
  - 若甲地到乙地有若干条公路，哪条公路最短或花费最小
- 考虑到交通图的有向性，只考虑带权有向图。
  - 源点：路径上的第1个顶点；终点：路径上最后1个顶点

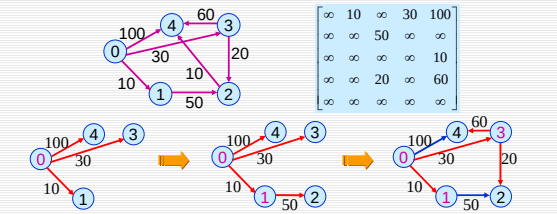
单源点最短路径

- **单源点最短路径问题**
  - 问题描述：给定带权有向图G＝(V, E)和源点v∈V，求从v到G中其余各顶点的最短路径。
- 其他最短路径问题：均可用单源最短路径算法予以解决
  - 单目标最短路径问题：找出图中每一顶点v到某指定顶点u的最短路径(只需将图中每条边反向)
  - 单顶点对间最短路径问题：对于某对顶点u和v，找出从u到v的一条最短路径。
  - 所有顶点对间最短路径问题：对图中每对顶点u和v，找出u到v的最短路径问题(每个顶点作为源点调用一次算法)

2. 迪杰斯拉特(Dijkstra)算法

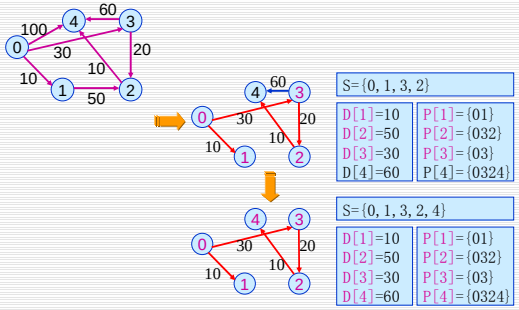
- 按路径长度递增次序产生最短路径算法
- 方法：设置辅助向量path记录路径，dist记录最短路径长度，s存放源点和已生成的终点
  - dist[i]初值：若从v_0到v_i有弧，则dist[i]为弧上权值，否则为∞。
  - 第一条最短路径是：从初态的dist中选一条值最小的dist[j]，对应的顶点v_j加入s，此路径为(v_0, v_j)，显然
    $$dist[j]=Min\{dist[i] \mid v_i \in V\}$$
  - 从v_0到其它顶点的最短路径有可能通过v_j得到改变。下一条次短路径是终点v_k,则这条路径或者是(v, v_k)，或者是(v, v_j, v_k)。它的长度或者是从v到v_k的权值，或者是dist[j]与从v_j到v_k的权值之和，因此要修改path，dist的值

Dijkstra算法举例



S={0}	S={0, 1}	S={0, 1, 3}
D[1]=10 P[1]={01} D[2]=∞ P[2]={} D[3]=30 P[3]={03} D[4]=100 P[4]={04}	D[1]=10 P[1]={01} D[2]=60 P[2]={012} D[3]=30 P[3]={03} D[4]=100 P[4]={04}	D[1]=10 P[1]={01} D[2]=50 P[2]={032} D[3]=30 P[3]={03} D[4]=90 P[4]={034}

Dijkstra算法举例(.续)



S={0, 1, 3, 2}	
D[1]=10 D[2]=50 D[3]=30 D[4]=60	P[1]={01} P[2]={032} P[3]={03} P[4]={0324}

S={0, 1, 3, 2, 4}	
D[1]=10 D[2]=50 D[3]=30 D[4]=60	P[1]={01} P[2]={032} P[3]={03} P[4]={0324}