Ch7 Graphs and Graph Travesral

7.1 Introduction

- 图的遍历
- 利用图的遍历方法解决问题：
  - 连通分量
  - 深（广）度优先搜索生成树
  - 二连通分量
  - …

图也称为网络，应用广泛：计算机网络，交通网络，社交网络，基因调控网络，蛋白质交互作用网络，代谢网络，……

7.2 Definitions and Representation

- 定义:有向图，无向图，路径，回路，强连通分量、连通分量…
- 带权图
- 图的存储方式—邻接矩阵，邻接表

Ch7.3 Traversing Graphs

- 从图中某个顶点出发访遍图中所有顶点，并且使图中的每个顶点仅被访问一次的过程:

1. 深度优先搜索
2. 广度优先搜索

在图的深度优先搜索过程中，每个顶点的状态有三种:

- 未被访问(undiscoverd)，
- 已经访问但从他出发的深度优先搜索尚未结束(discoverd)，
- 已经访问且从他出发的深度优先搜索已经结束(finished)

- **深度优先搜索生成树**
- **深度优先搜索生成森林**

实现方法

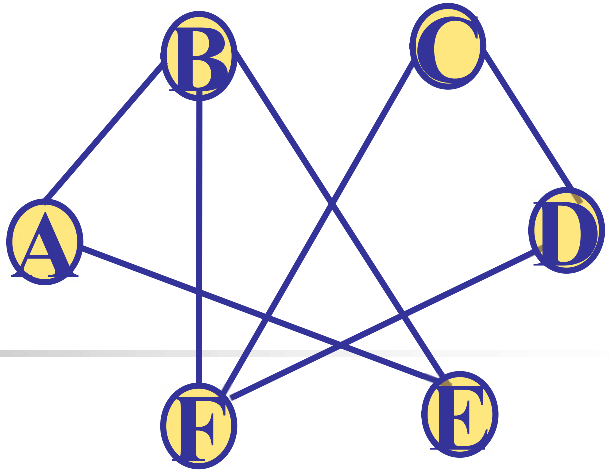
```
void dfs (G, v) { // 从顶点v出发，深度优先搜索遍历图 G
    mark v as discoverd;
    for each vertex w such that edge vw is in G
        if w is undiscoverd dfs(G,w);
    mark v as finished;
 } // DFS
void dfsSweep(G) {// 深度优先搜索遍历图 G
     initialize all vertices of G to undiscovered;
     for each vertex v∈G
         if v is undiscoverd dfs(G, v);}
```

邻接矩阵存储----**O(n+n^2)**
邻接表存储----**O(n+e)**

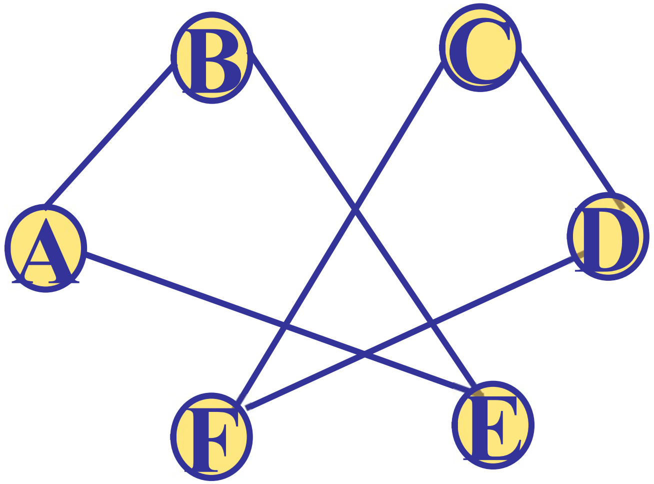
7.4 图搜索的应用--连通分量

- 利用深度优先搜索和广度优先搜索
  - 判断无向图是否为连通图?
  - 几个连通分量?
  - 确定每个连通分量的顶点
  - ……
- connected component的确定也可采用并查集

若图G中任意两个顶点之间都有路径相通，则称此图为连通图



若无向图为非连通图，则图中各个极大连通子图称作此图的连通分量。



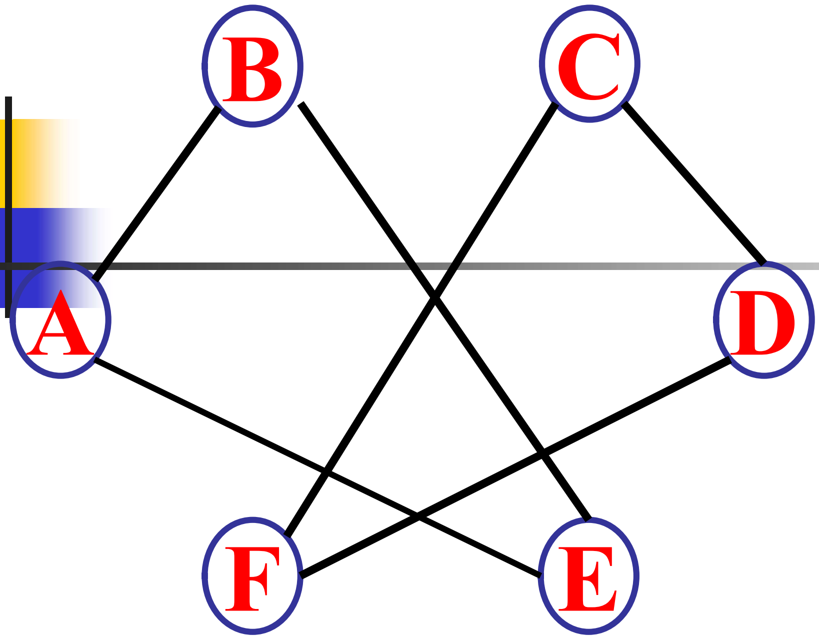
深（广）度优先搜索过程中选几次出发点，就有几个连通分量。通过表示顶点的访问<u>起源于哪个出发点</u>(或源于第几个出发点)，可以确定顶点属于哪个连通分量

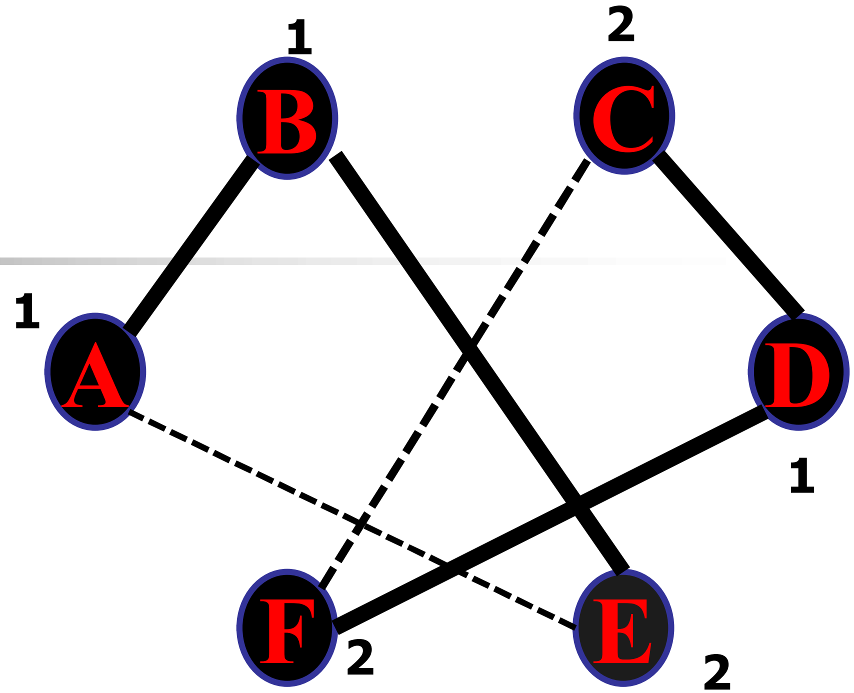
书中将在图的深（广）度优先搜索过程中，每个顶点的三种状态分别用三种颜色表示：

white----undiscoverd,

gray----discoverd,

black----finished

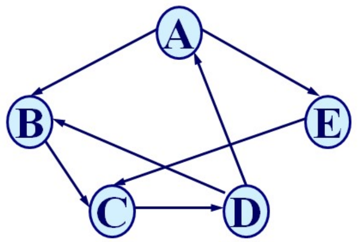




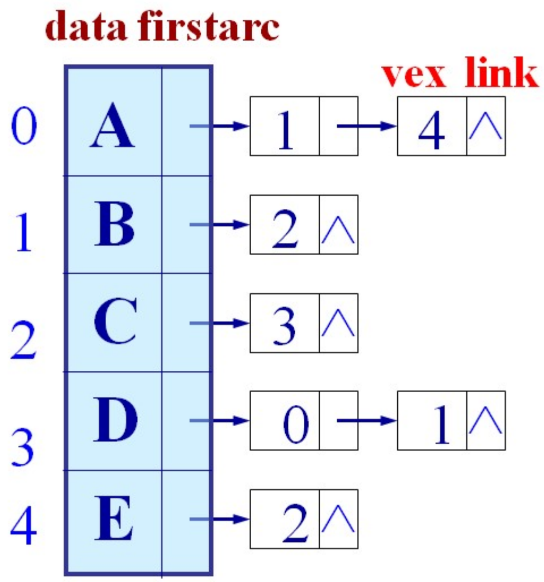
- **A,B,E,C,D,F**

  选了2个出发点，有两个连通分量，非连通图

  A，B，E位于同一连通分量
  C，D，F位于同一连通分量




```
typedef struct ArcNode {
  int       vex;  // 该弧所指向的顶点的位置
  struct ArcNode  *link; // 指向下一条弧的指针
  ……
} ArcNode;
typedef struct VNode {
  VertexType  data;   // 顶点信息
  ArcNode  *firstarc; // 指向第一条依附该顶点的弧
  } VNode;
 typedef struct {
     VNode    arcs[MAXSIZE];
     int      vexnum, arcnum;
     int      kind; //图的类型
  } Graphs;
```



```
void connectedComponents(Graphs G, int ccNum, int cc[ ])
{  for (v=0; v<G.vexnum; ++v)
     color[v] = white;
     ccNum=0;
   for (v=0; v<G.vexnum; ++v)
     if (color[v]==white)
          {ccNum++;
           ccDFS(G, color,  v, ccNum, cc);}
}

void ccDFS(Graphs G, int color[ ],  int v, int ccNum, int cc[ ]);
{   color[v] = gray;   cc[v]=ccNum;
    for(p=G.arcs[v].firstarc; p!=NULL; p=p->link)
    {    w=p->vex;
       if (color[w]==white) ccDFS(G, color,  w, ccNum, cc);
    }
    color[v]=black;
}
```

分析：时间----$\Theta(n+e)$

空间----$\Theta(n+e)$，$\Theta(n)$

7.4 图搜索的应用--有向无环图

- 实际应用----活动安排----有环意味着“死锁”
- 解决问题的算法的效率

拓扑排序

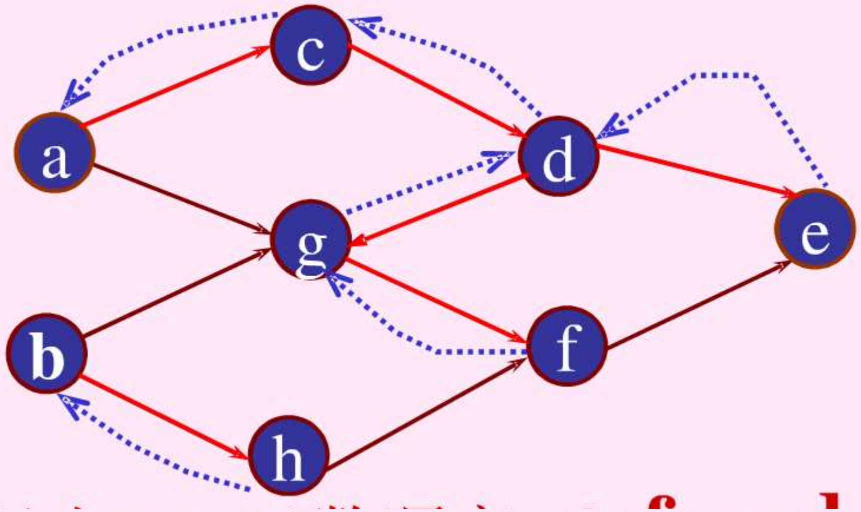
- 拓扑排序定义:
  - 图G=(V,E)是n个顶点的有向图。图G的拓扑排序（toplogical order）是指给图中的每一顶点赋予1,⋯,n的不同整数，使得如果vw∈E,则v的拓扑序号小于w的。
  - reverse toplogical order
- 定理：若有向图G中存在回路，则G不存在拓扑排序

Definition 7.16 Topological order

Let $G = (V, E)$ be a directed graph with n vertices. A *topological order* for G is an assignment of distinct integers $1, \ldots, n$ to the vertices of V, called their *topological numbers*, such that, for every edge $vw \in E$, the topological number of v is less than the topological number of w. A *reverse topological order* is similar except that for every edge $vw \in E$ the topological number of v is greater than the topological number of w. ■

深度优先搜索构造拓扑排序

对有向无环图利用深度优先搜索进行拓扑排序。



最先退出**DFS**函数的顶点是出度为零的顶点，为拓扑排序序列中最后一个顶点。

因此，按退出**DFS**函数的先后记录下来的顶点序列即为逆向的拓扑排序序列。

```
void dfsSweep(G) {
      color all vertices to white;
       topoNum=G.vexnum;
      for each vertex v∈G
           if v is white dfs(G, v);}
```

```
void dfs (G, v) {
    color v as gray;
    for each vertex w such that edge vw is in G
           if w is white dfs(G,w);
     topo[v]=topoNum;topoNum--;
    color v as black;
 } // DFS
```