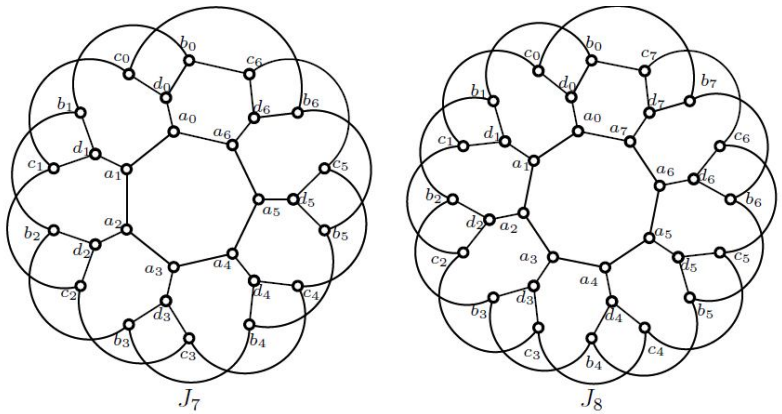
7.3 图的遍历

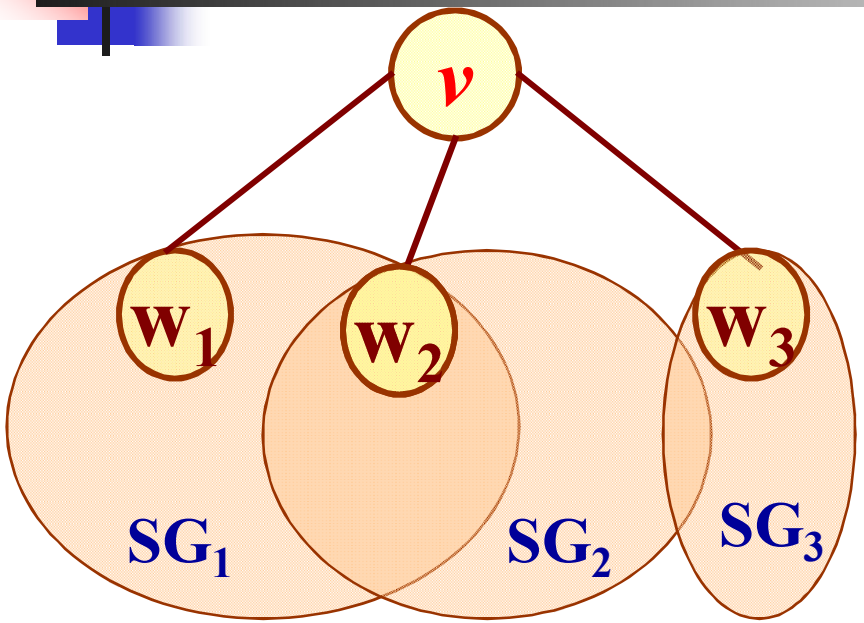
- 从图中某个顶点出发访遍图中其余顶点，并且使图中的每个顶点仅被访问一次的过程。
- 遍历策略：深度优先搜索和广度优先搜索
- 遍历应用举例：图的连通性等



7.3 图的遍历--深度优先搜索

- 从图中某个未被访问的顶点v出发，访问此顶点，然后依次从v的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和v有路径相通的顶点都被访问到。
- 若图中仍存在未被访问的顶点，则从中选择一点做出发点，重复上述过程，直至图中所有的顶点都被访问到。

7.3 图的遍历--深度优先搜索



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。

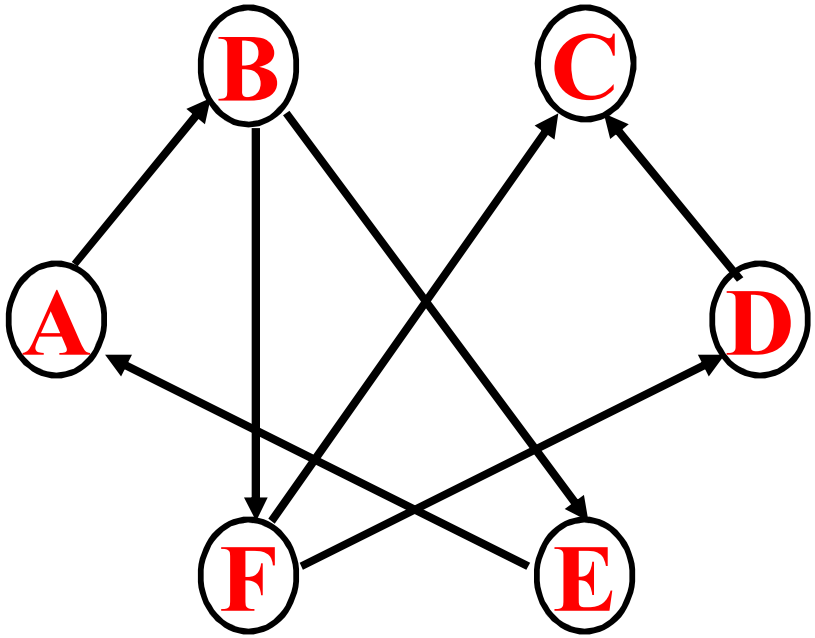
访问顶点 *v*：

for (W_1、W_2、W_3)

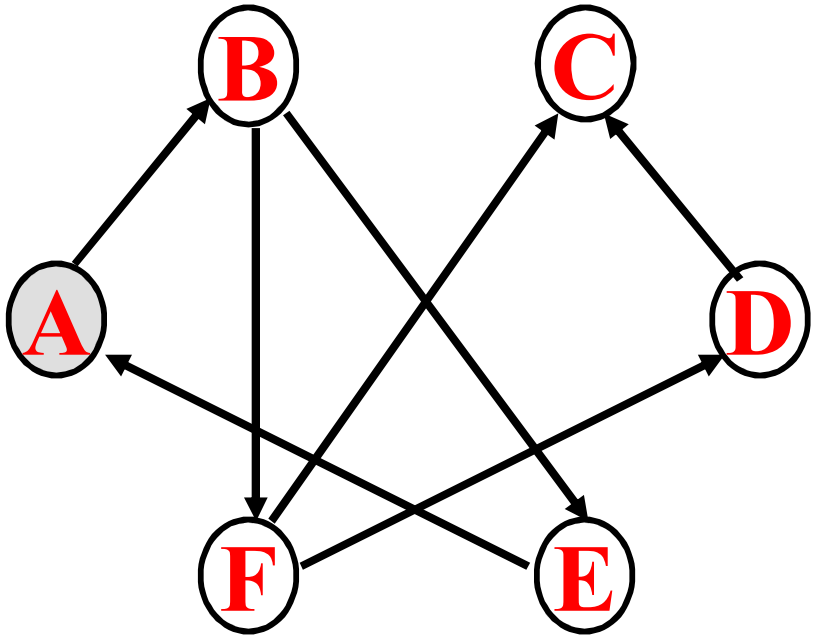
若该邻接点W未被访问，

则从它出发进行深度优先搜索遍历。

深度优先搜索

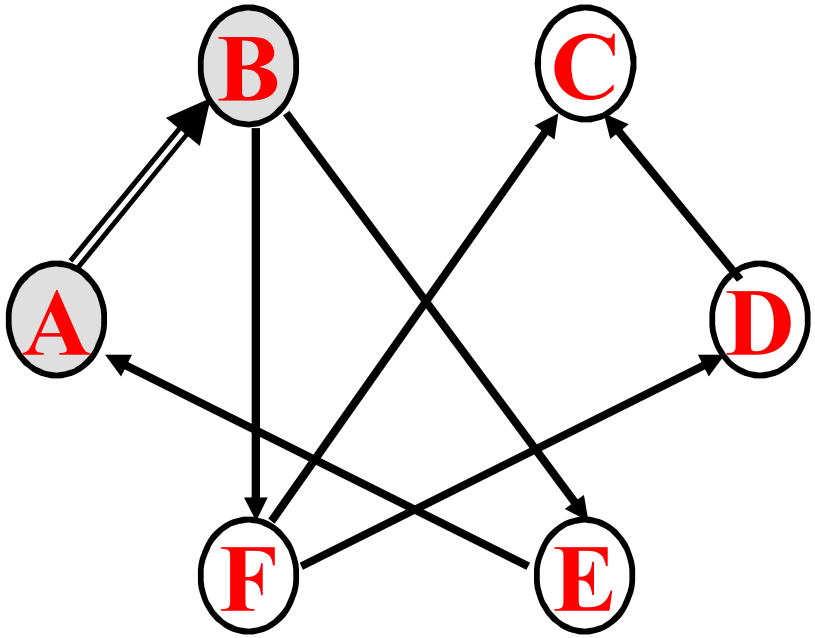


深度优先搜索



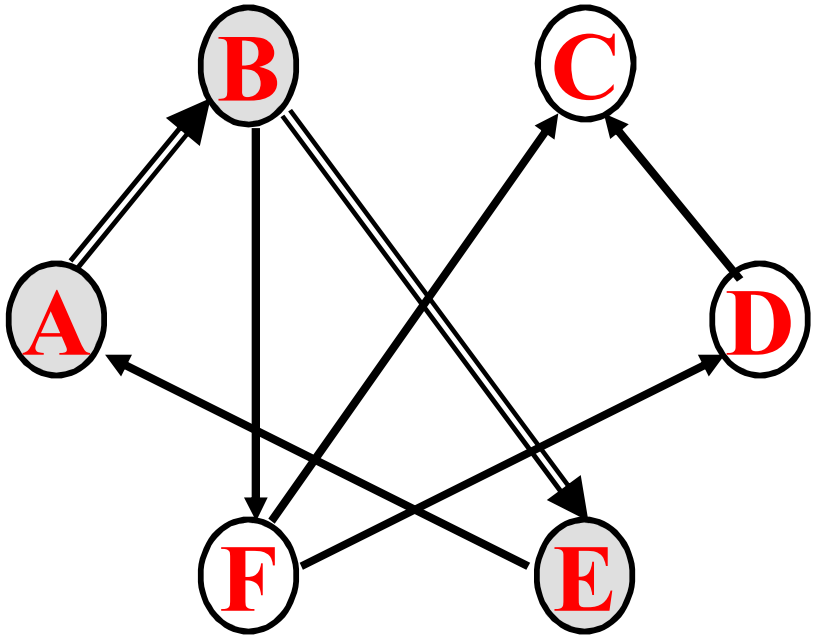
- A

深度优先搜索



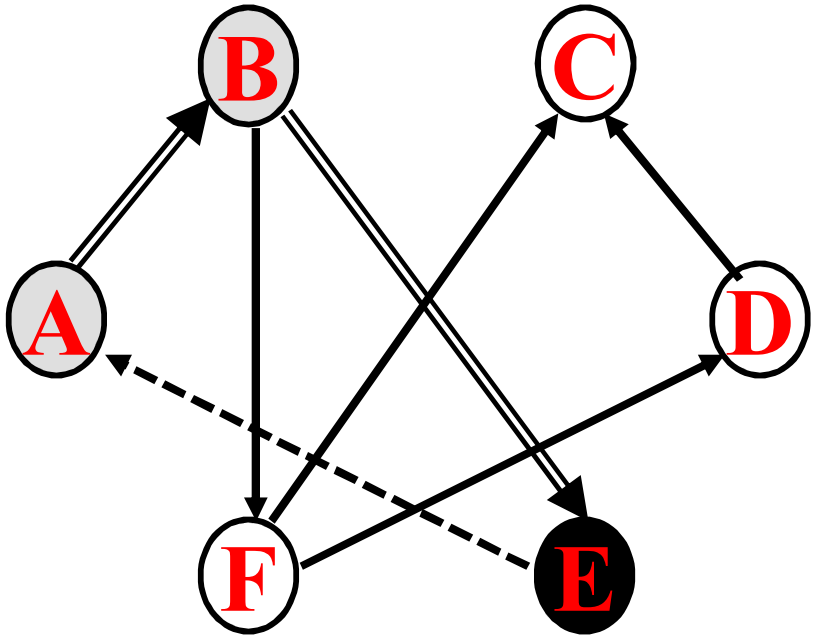
- A,B,

深度优先搜索



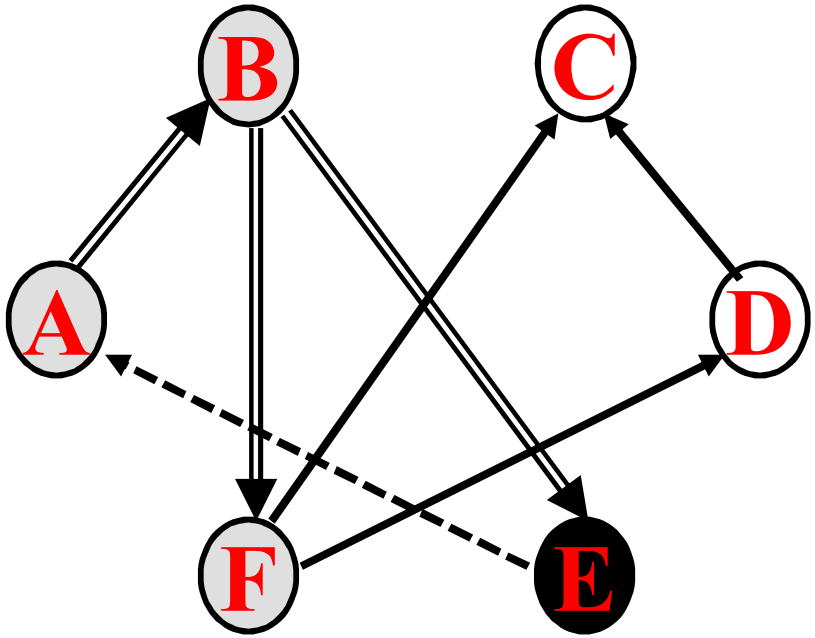
- A,B,E

深度优先搜索



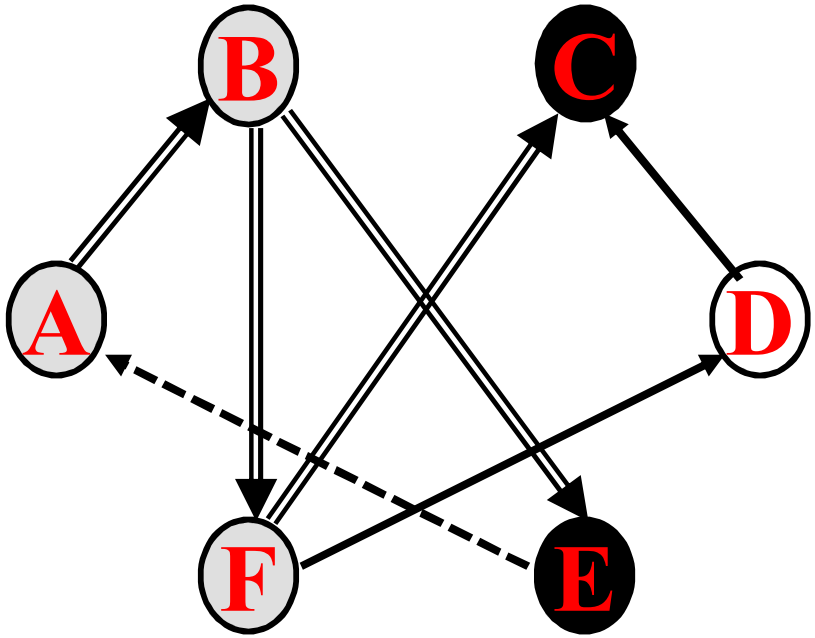
- A,B,E

深度优先搜索



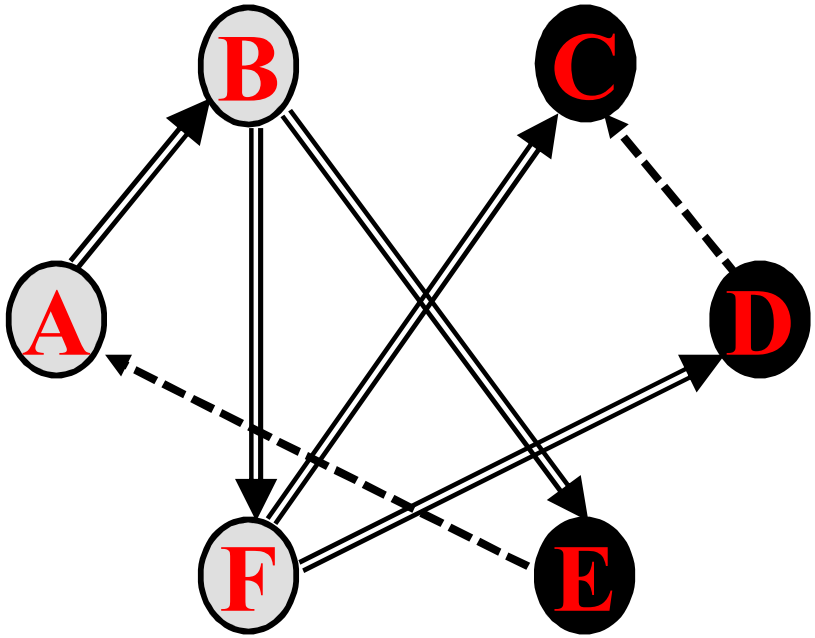
- A,B,E,F

深度优先搜索



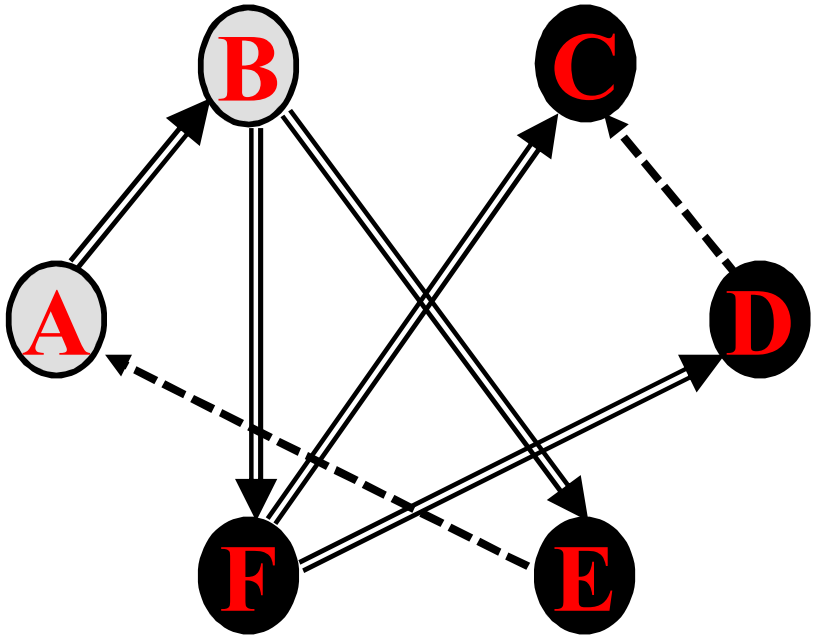
- **A,B,E,F,C**

深度优先搜索



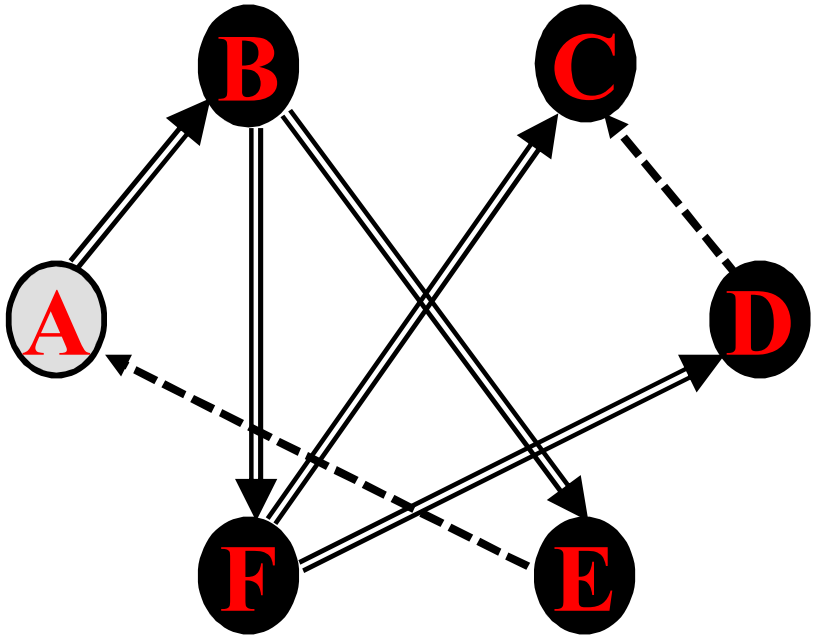
- **A,B,E,F,C,D**

深度优先搜索



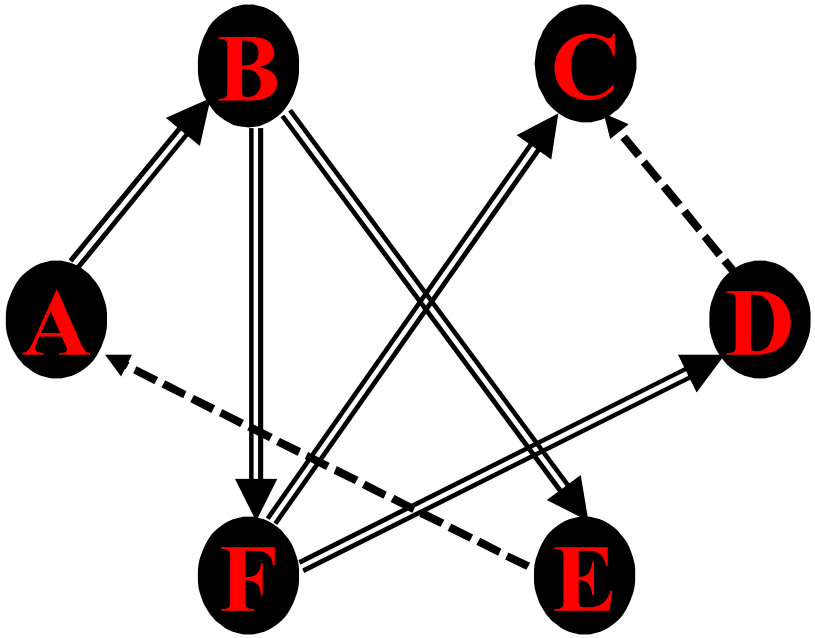
- A,B,E,F,C,D

深度优先搜索



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

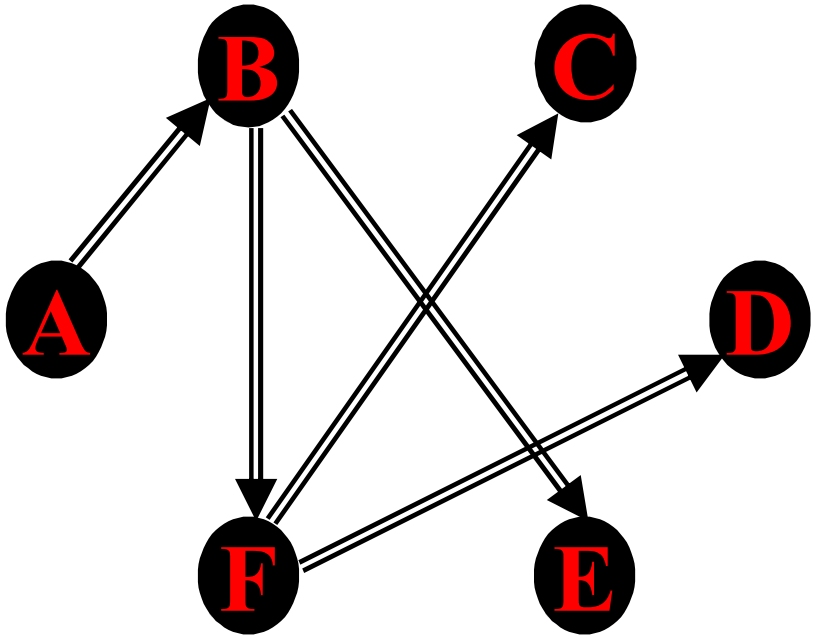
深度优先搜索



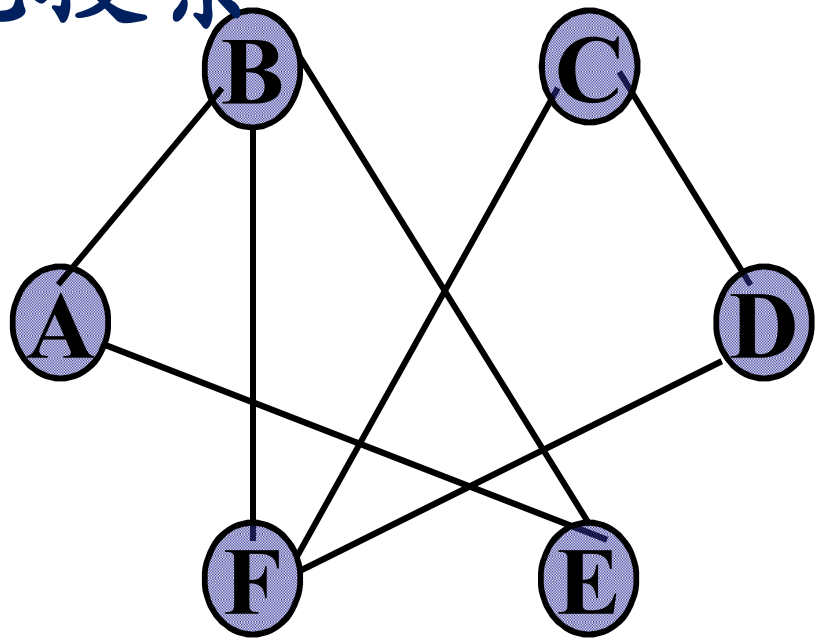
- A,B,E,F,C,D

深度优先搜索

- **深度优先搜索生成树**：访问时经过的顶点和边构成的子图
- **深度优先搜索生成森林**：若选用多个出发点做深度优先搜索，会产生多棵深度优先搜索生成树—构成深度优先搜索生成森林

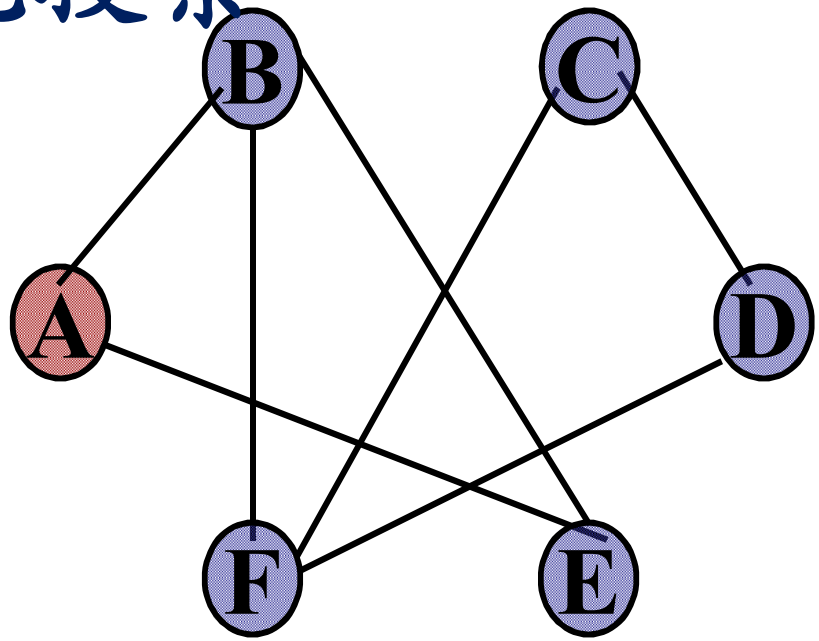


深度优先搜索



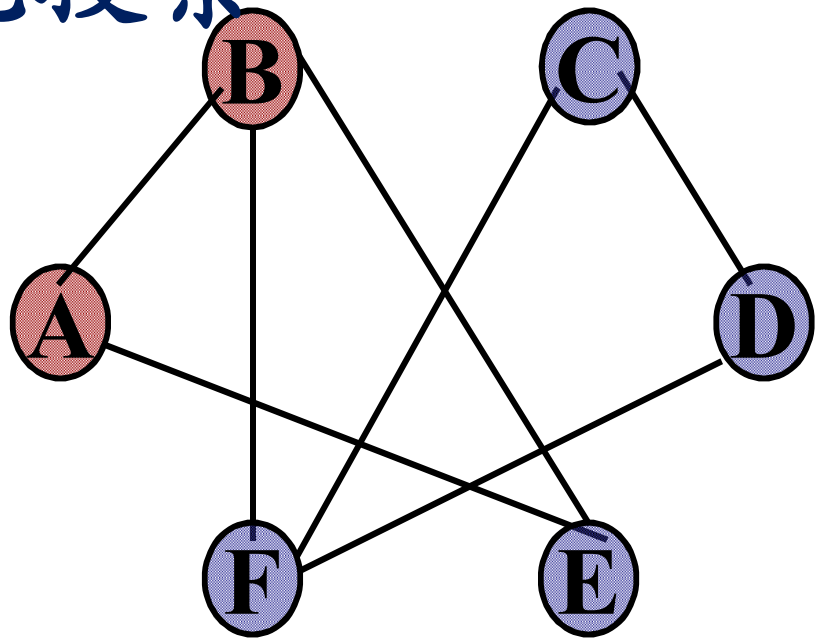
深度优先搜索

A



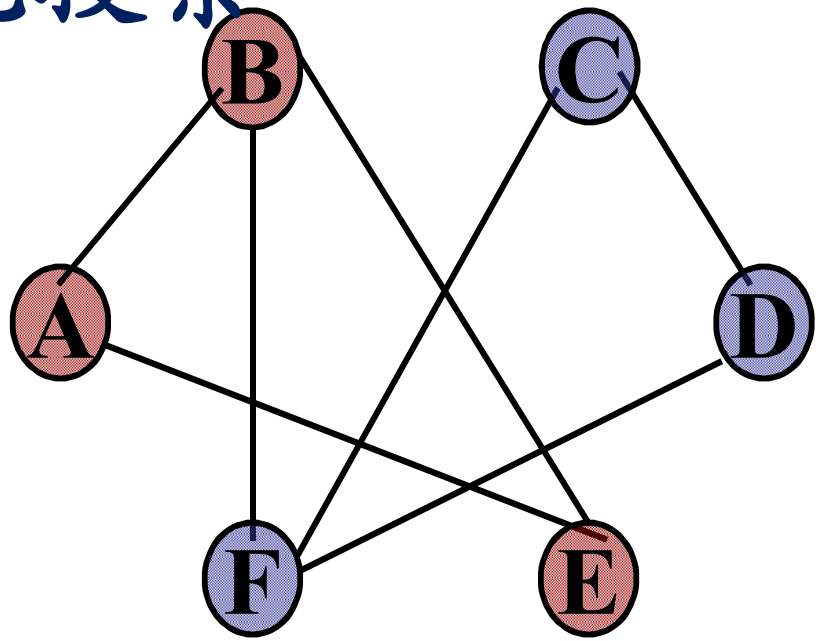
深度优先搜索

AB



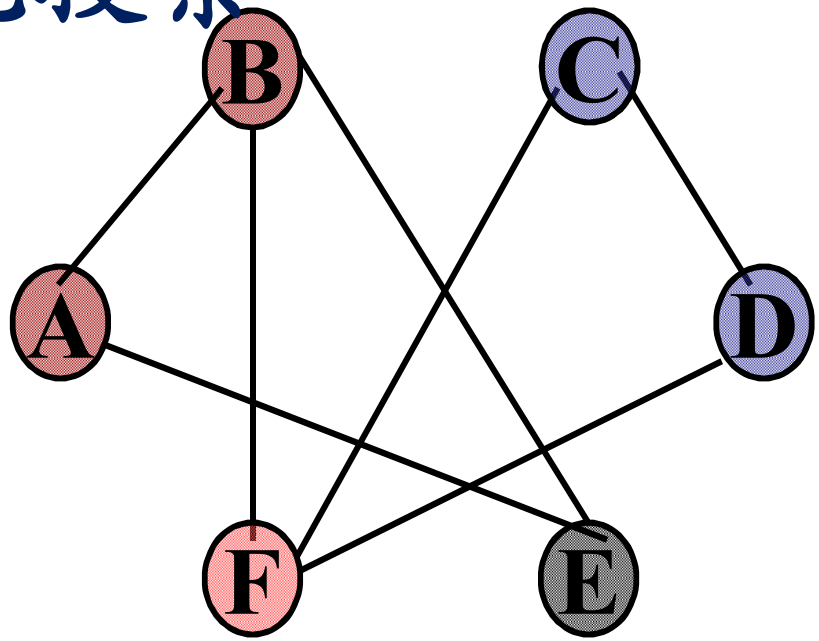
深度优先搜索

ABE



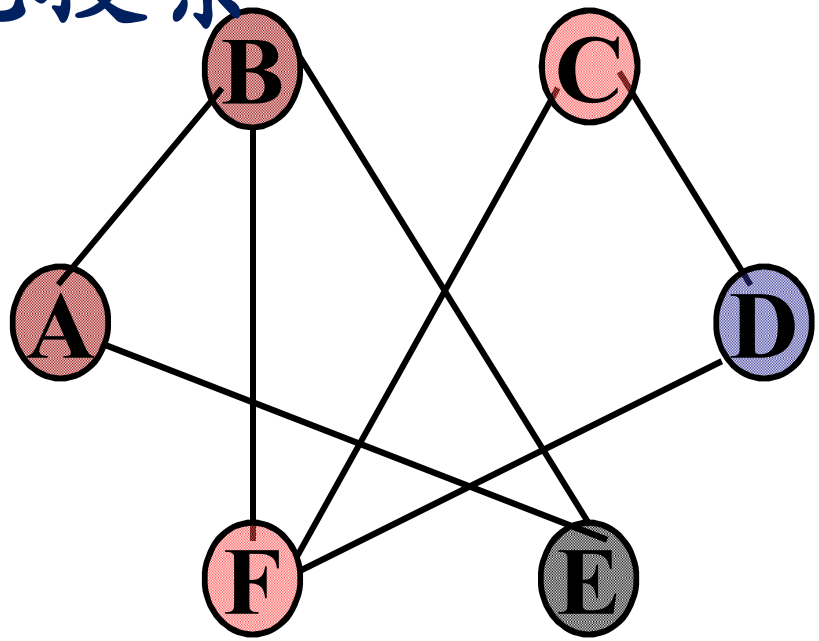
深度优先搜索

ABEF



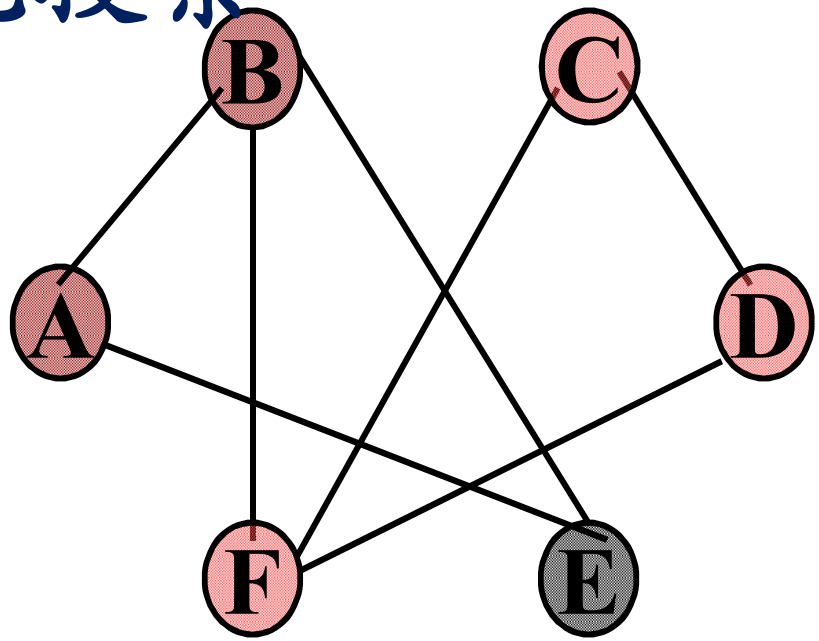
深度优先搜索

ABEFC



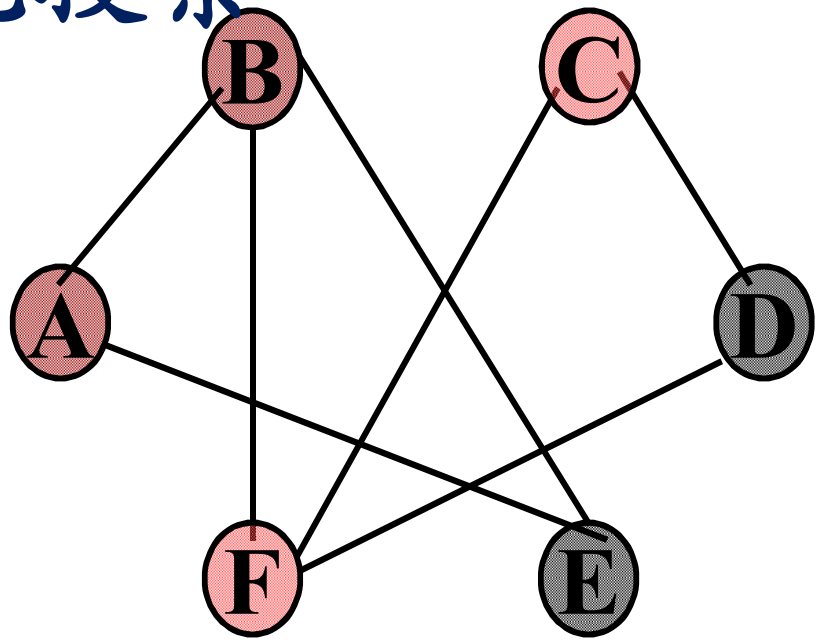
深度优先搜索

ABEFCD



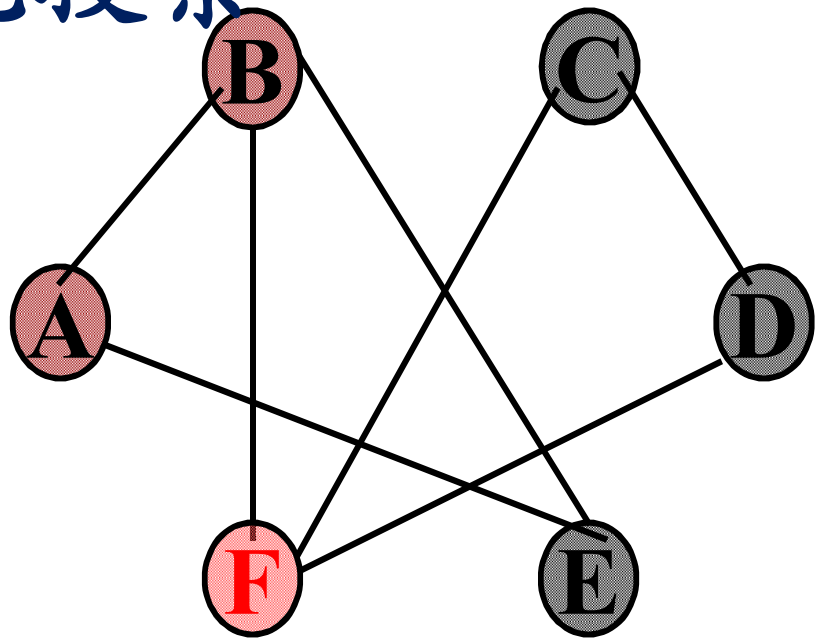
深度优先搜索

ABEFCD



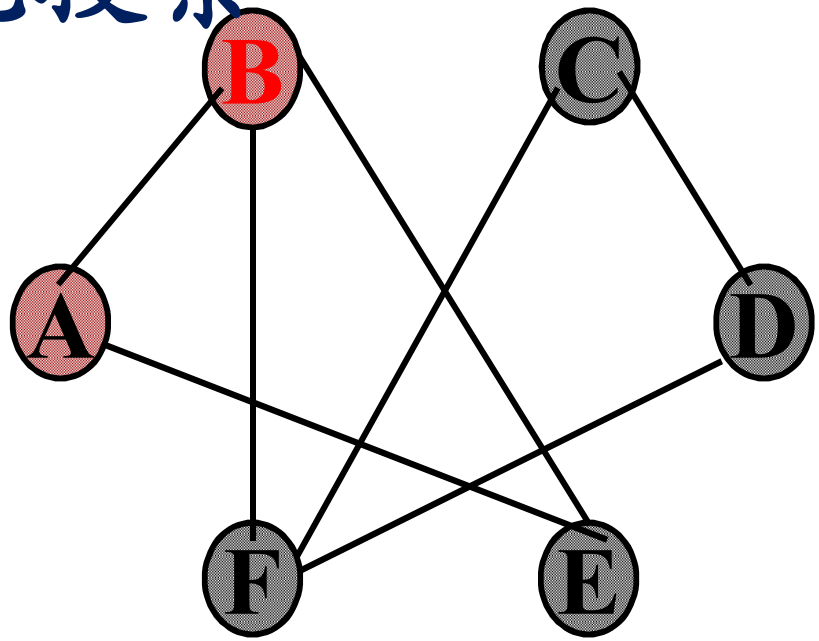
深度优先搜索

ABEFCD



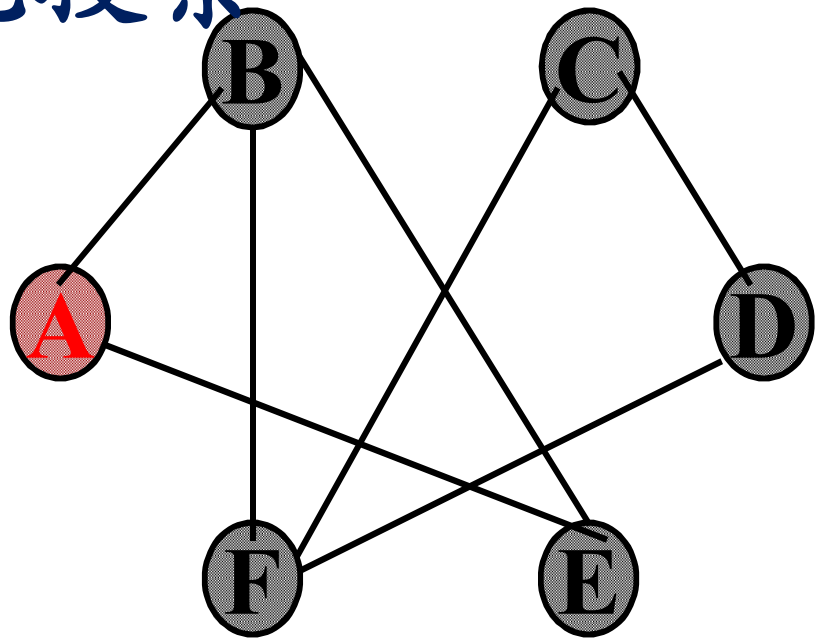
深度优先搜索

ABEFCD



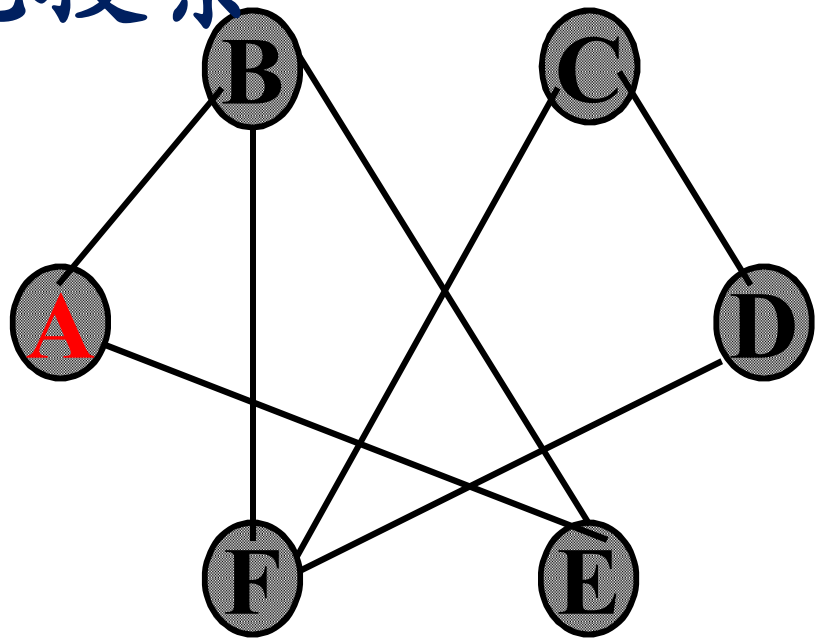
深度优先搜索

ABEFCD



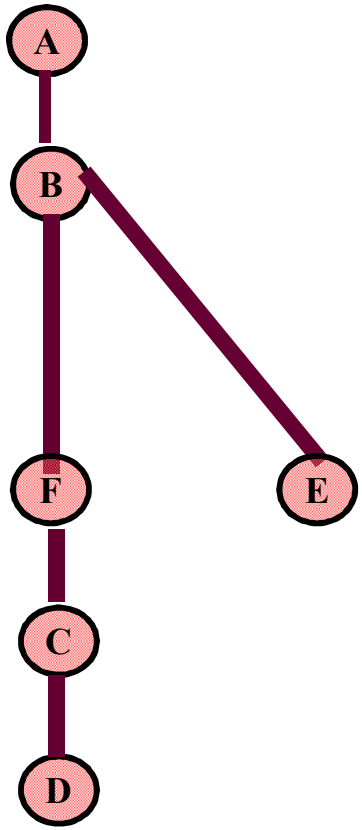
深度优先搜索

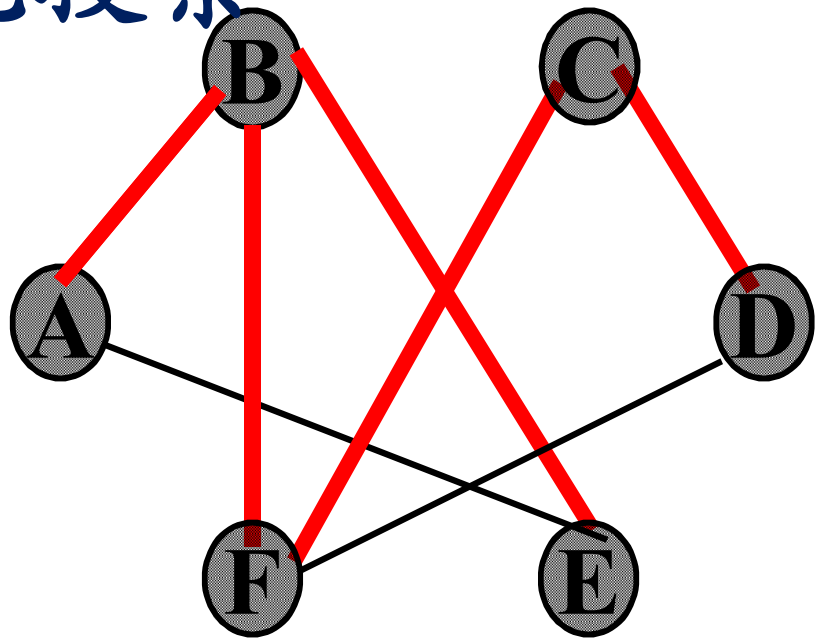
ABEFCD



深度优先搜索

ABEFCD

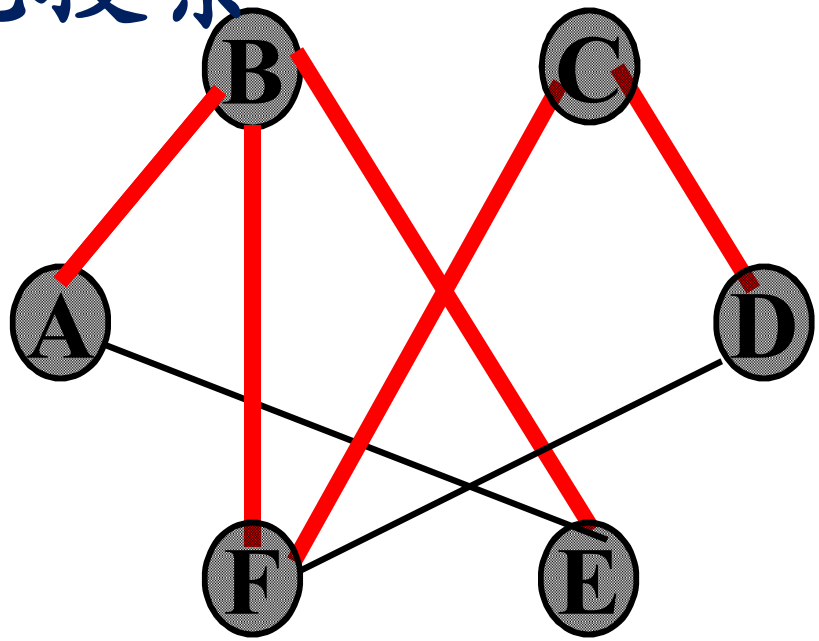


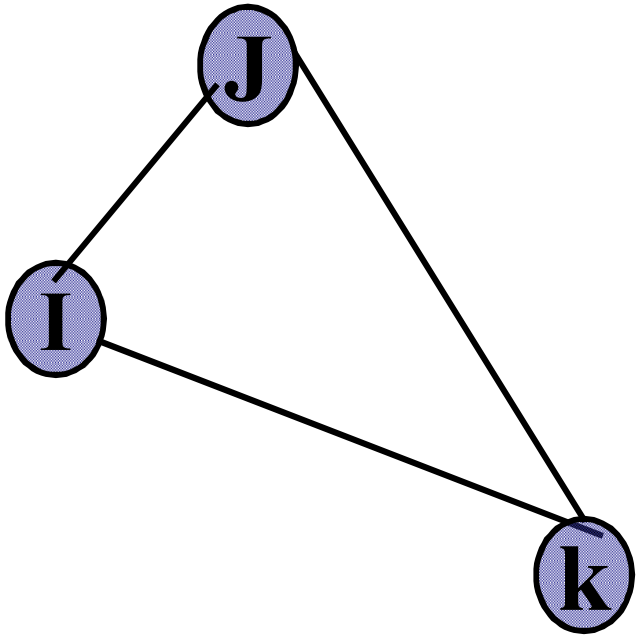


深度优先搜索生成树：访问时经过的顶点和边构成的子图

深度优先搜索

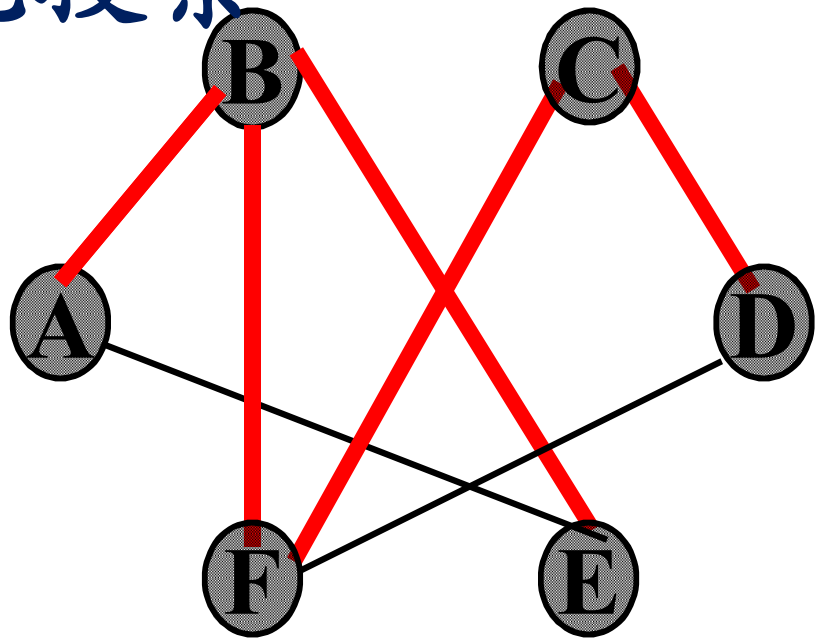
ABEFCD

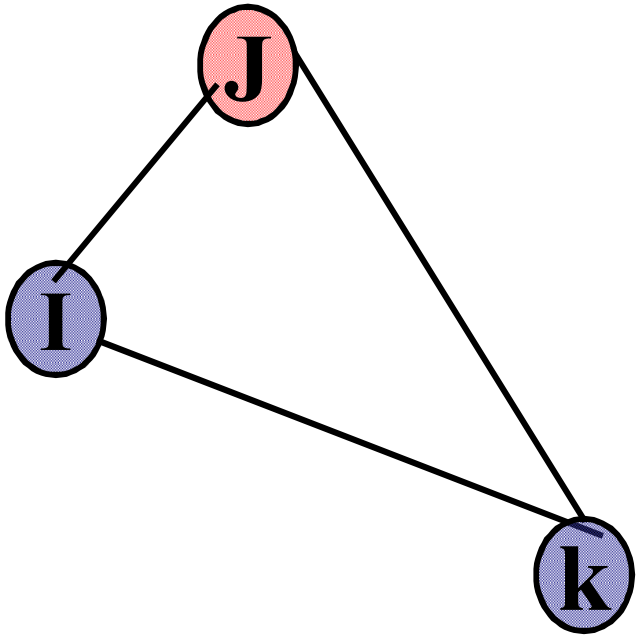




深度优先搜索

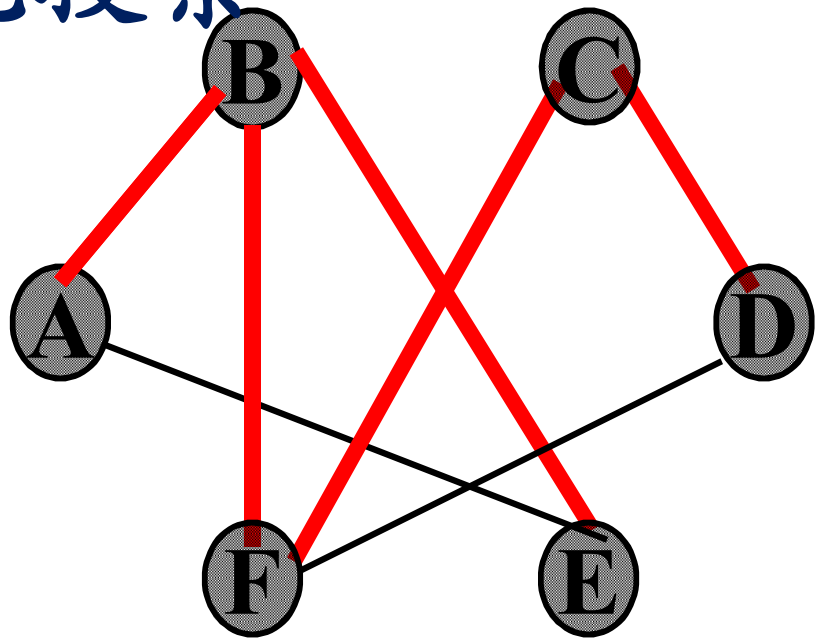
ABEFCDJ

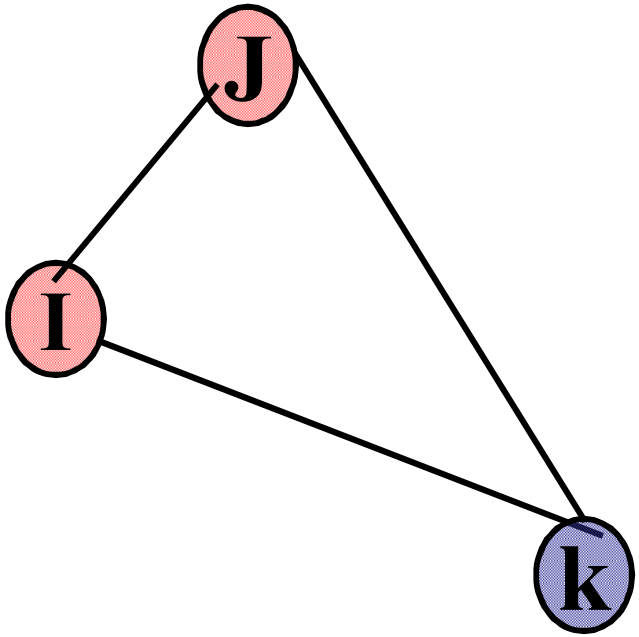




深度优先搜索

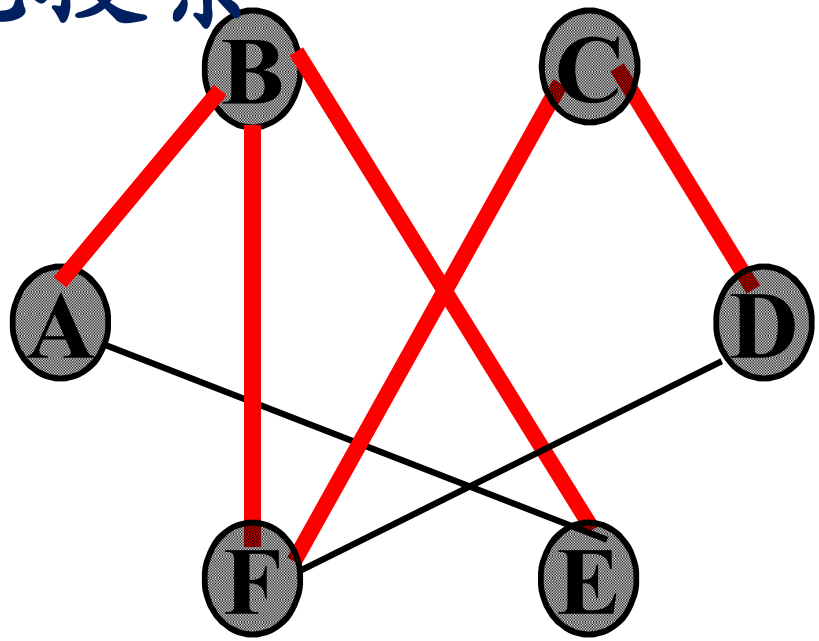
ABEFCDJI

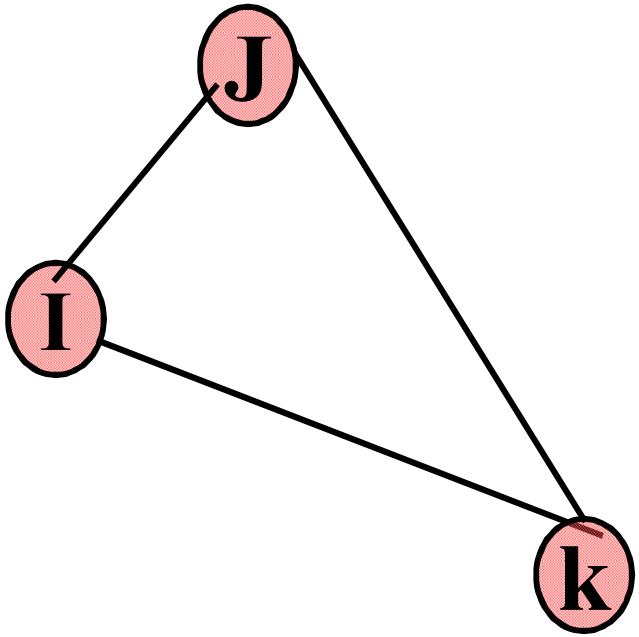




深度优先搜索

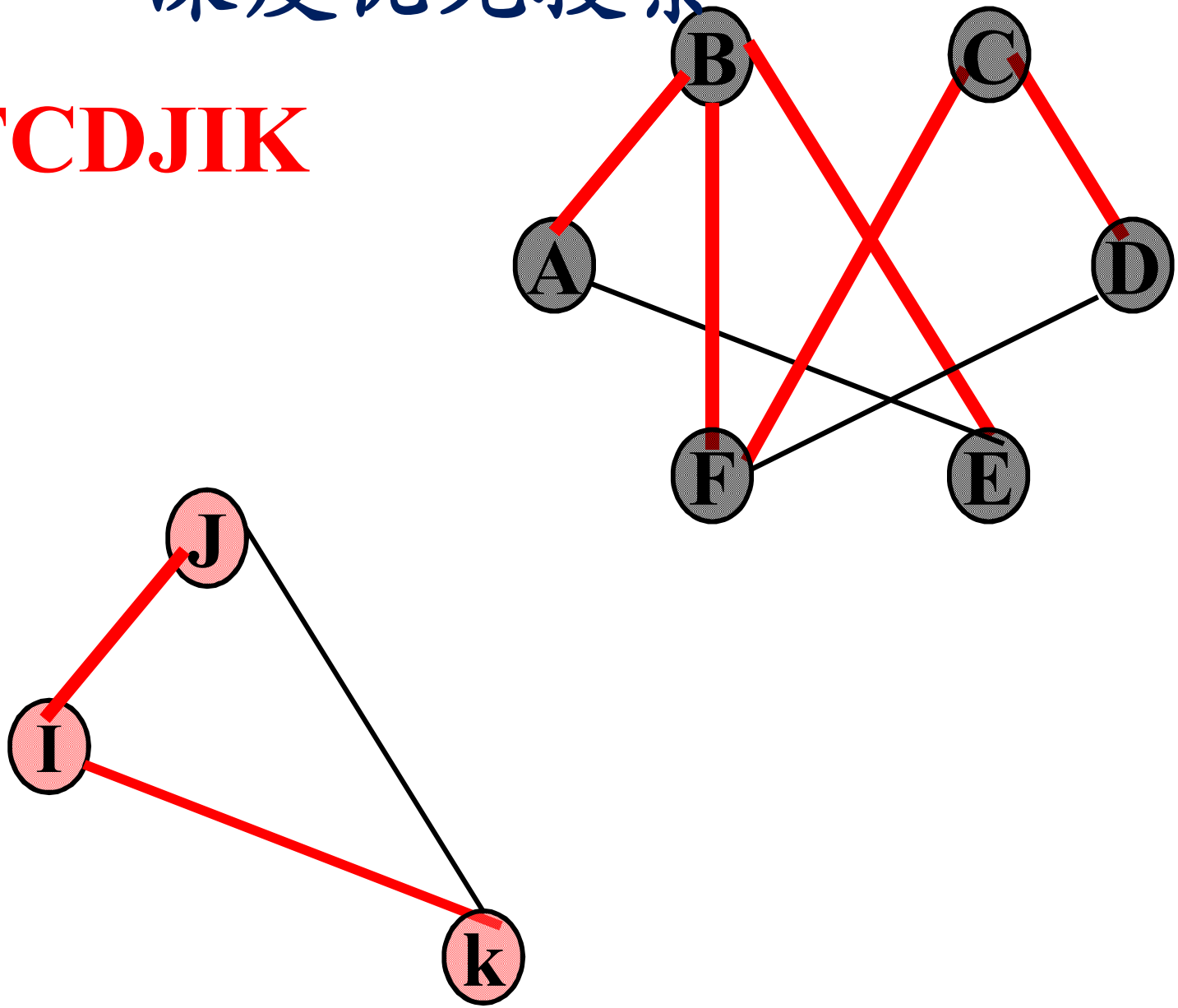
ABEFCDJIK





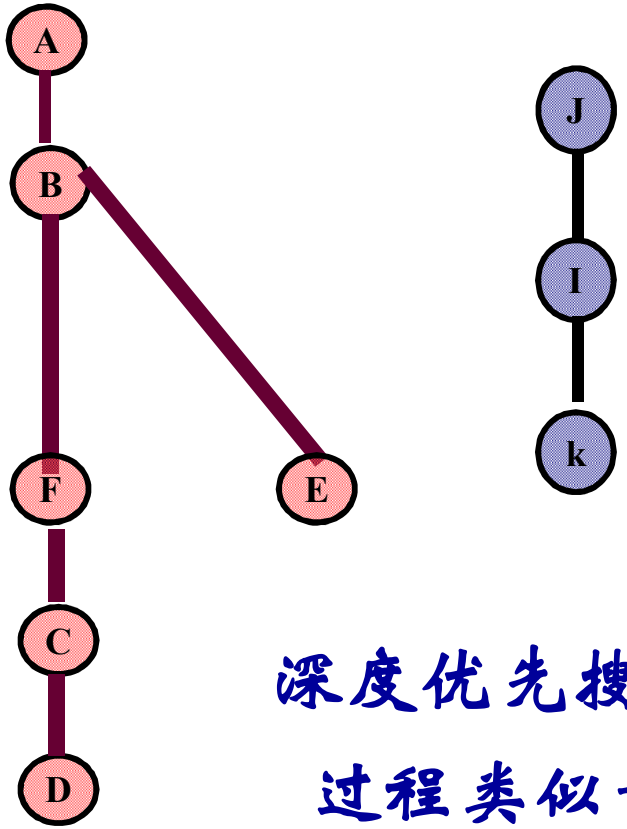
深度优先搜索

ABEFCDJIK



非连通图的深度优先搜索

深度优先搜索生成森林



判断无向图是否连通？

若从无向图中任一点出发能访问到图中所有顶点，则该图为连通图

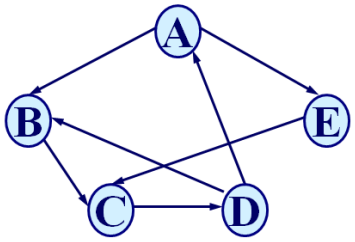
判断有向图是否强连通？

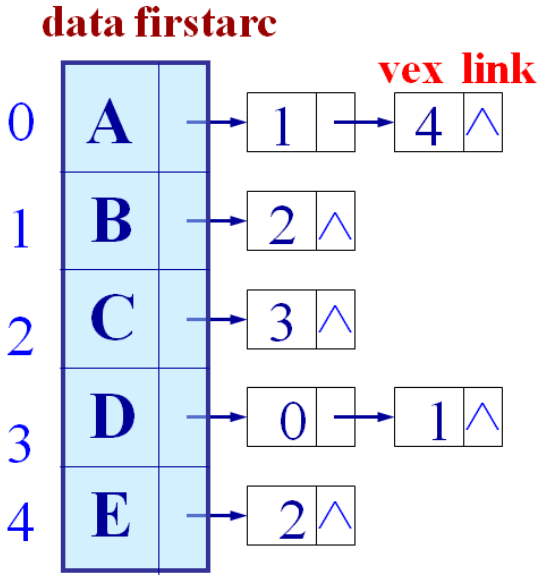
若从有向图中每一点出发能访问到图中所有顶点，则该图为强连通图

深度优先搜索遍历连通图的过程类似于树的先根遍历

7.3 图的遍历--深度优先搜索算法实现

- 图的存储？邻接矩阵和邻接表均可以
- 如何判别v的邻接点是否被访问？
  - 解决的办法：为每个顶点设立一个“访问标志”，设一维数组visited[]，visited[w]=1表示顶点w已经被访问；visited[w]=0表示顶点w尚未被访问。





以邻接表为例实现图的深度优先搜索

```
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
  InfoType   *info;   // 该弧相关信息的指针
} 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    arc[MAXSIZE];
     int      vexnum, arcnum;
     int      kind; //图的类型
  } Graphs;
```

7.3 图的遍历--深度优先搜索算法实现

```
void DFS(Graphs G, int v) {
  // 从顶点v出发，深度优先搜索遍历图G
   visited[v] = 1;   printf("%d",v);
   for(p=G.arc[v].firstarc; p!=NULL; p=p->link)
  {    w=p->vex;
       if (!visited[w])  DFS(G, w);
   }
} // DFS
```

7.3 图的遍历--深度优先搜索算法实现

```
void DFSTraverse(Graphs G)
{  // 对图 G 作深度优先遍历
   for (v=0; v<G.vexnum; ++v)
    visited[v] = 0; // 访问标志数组初始化
   for (v=0; v<G.vexnum; ++v)
    if (!visited[v])  DFS(G, v);
          // 对尚未访问的顶点调用DFS
}
```