

7.4 图的连通性问题

7.4.1 无向图的连通分量和生成树

7.4.3 最小生成树

7.4.1 无向图的连通分量和生成树

```
void DFSForest( Graph G, CSTree &T ) {  
    //建立无向图G的深度优先生成森林的(最左)孩子(右)兄弟链表T  
    T=NULL;  
    for ( v=0; v<G.vexnum; ++v )  
        visited[v]=FALSE;  
    for ( v=0; v<G.vexnum; ++v )  
        if (!visited[v]){  
            //第v顶点为新的生成树的根结点  
            p=(CSTree) malloc (sizeof ( CSNode ));    //分配根结点  
            *p={GetVex(G,v), NULL, NULL};            //给该结点赋值  
            if ( !T ) T = p;    //是第一棵生成树的根(T的根)  
            else q->nextsibling=p; //是其他生成树的根(前一棵的根的“兄弟”  
            q=p;    //q指示当前生成树的根  
            DFSForest(G, v, p);    //建立以p为根的生成树  
        }  
} // DFSForest
```

算法 7.7

```

void DFSTree( Graph G, CStree &T ) {
    //从第v个顶点出发深度优先遍历图G，建立以T为根的生成树。
    visited[v]=TRUE      ; first=TRUE;
    for ( w=FisrtAdjVex(G,v); w>=0; w=NextAdjVex(G,v,w) )
        if (!visited[w]){
            p=(CStree) malloc (sizeof ( CSNode ));    //分配孩子结点
            *p={GetVex(G,w), NULL, NULL};
            if ( first) {                //w是v的第一个未被访问的邻接顶点
                T->lchild=p; first=FALSE;        //是根的左孩子结点
            }//if
            else{                        //w是v的其他未被访问的邻接顶点
                q->nextsibling=p;    //是上一邻接顶点的有兄弟结点
            }//else
            q=p;
            DFSTree(G, w, q);
            //从第w个顶点出发深度优先遍历图G，建立子生成树q
        }//if
    }// DFSTree
}

```

算法 7.8

7.4.3 最小生成树

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。

算法一：（普里姆算法prim）

算法二：（克鲁斯卡尔算法kruskal）

算法利用了最小生成树的下列一种简称为MST的性质：假设 $N = (V, \{E\})$ 是一个连通网， U 是顶点集 V 的一个非空集合。若 (u, v) 是一条具有最小权值（代价）的边，其中 $u \in U, v \in V - U$ ，则必存在一棵包含边 (u, v) 的最小生成树。

可用反证法证明：假设网 N 任何一棵最小生成树都不含 (u, v) 。设 T 是一棵最小生成树，当将边 (u, v) 加入到 T 中时，由生成树定义， T 中必存在一条包含 (u, v) 的回路。另一方面， T 是生成树， T 中必存在另一条边 (u', v') ，其中 $u' \in U, v' \in V - U$ ，且 u 和 u' 之间， v 和 v' 之间均有路径相通，删去边 (u', v') ，便可消除回路，同时得到另一棵生成树 T' 。因为 (u, v) 的代价不高于 (u', v') ，则 T' 的代价亦不高于 T ， T' 是包含 (u, v) 的一棵最小生成树。由此和假设矛盾。

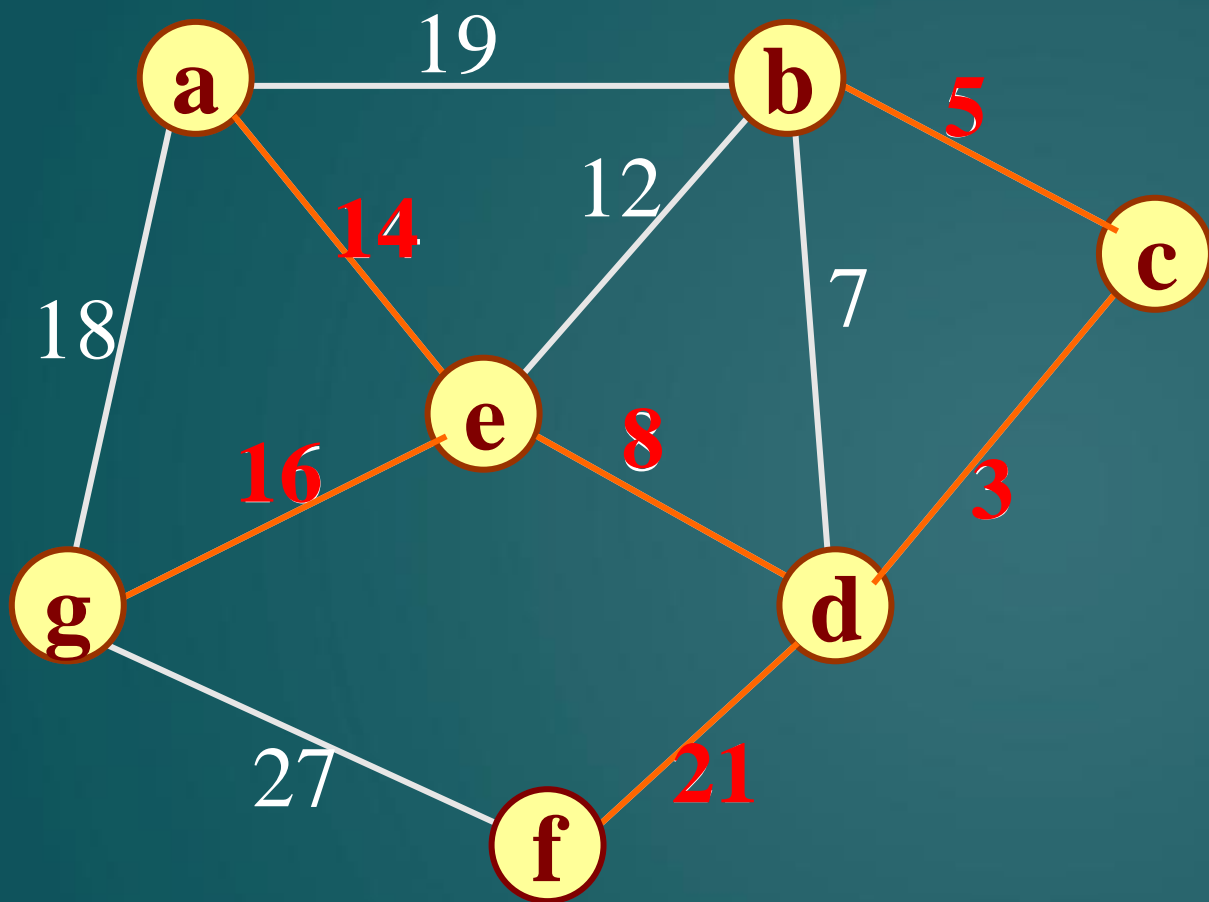
普里姆算法的基本思想:

取图中任意一个顶点 v 作为生成树的根，之后往生成树上添加新的顶点 w 。**在添加的顶点 w 和已经在生成树上的顶点 v 之间必定存在一条边，并且该边的权值在所有连通顶点 v 和 w 之间的边中取值最小。**之后继续往生成树上添加顶点，直至生成树上含有 $n-1$ 条边为止。

普里姆算法:

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 $U = \{u_0\} (u_0 \in V, TE = \{\})$ 开始，重复执行下述操作：在所有 $u \in U, v \in V - U$ 的边 $(u, v) \in E$ 中找一条代价最小的边 (u_0, v_0) 并入集合，同时 v_0 并入 U ，直至 $U = V$ 为止。此时 TE 中必有 $n-1$ 条边，则 $T = (V, \{TE\})$ 为 N 的最小生成树。

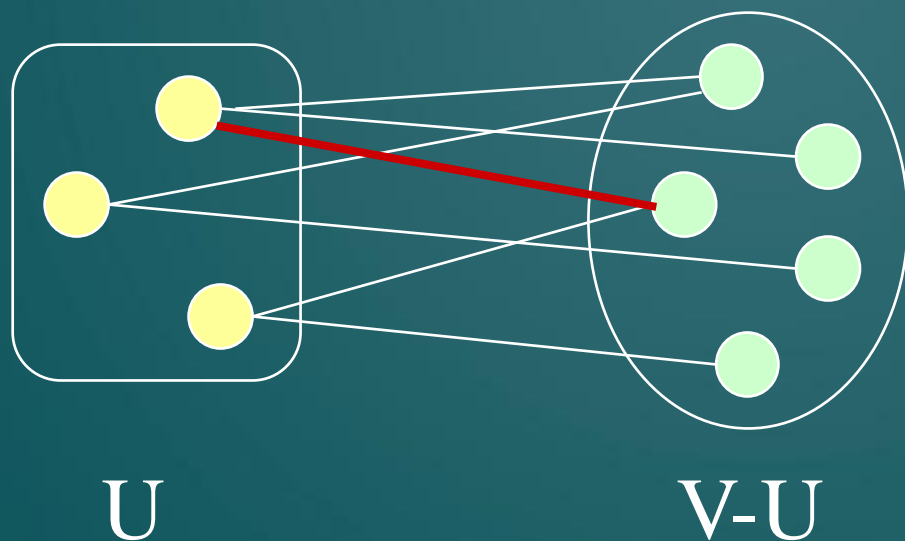
例如:

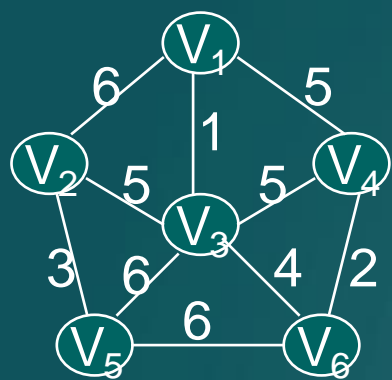


所得生成树权值和 = $14+8+3+5+16+21 = 67$

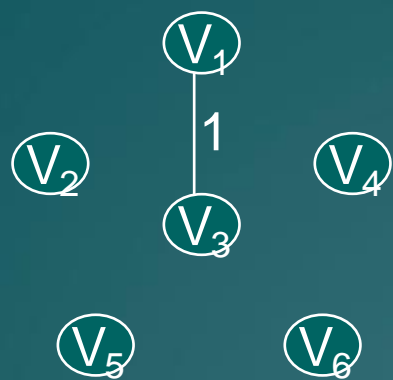
一般情况下所添加的顶点应满足下列条件:

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 和**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。

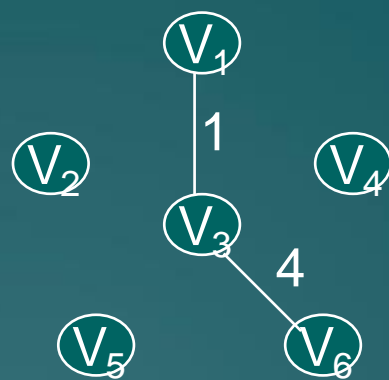




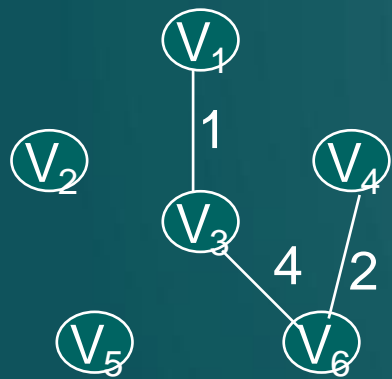
(a)



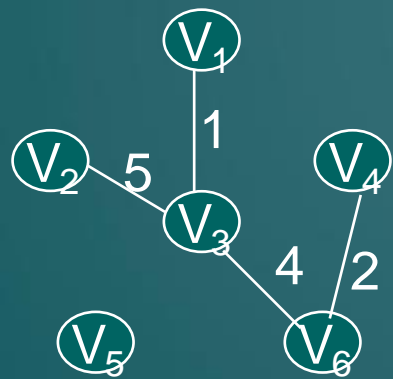
(b)



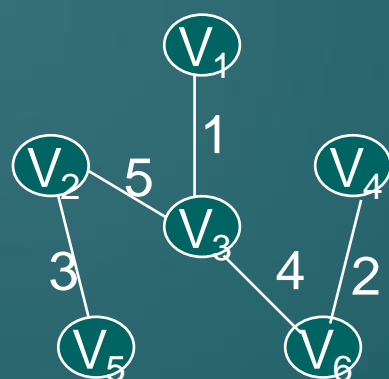
(c)



(d)



(e)



(f)

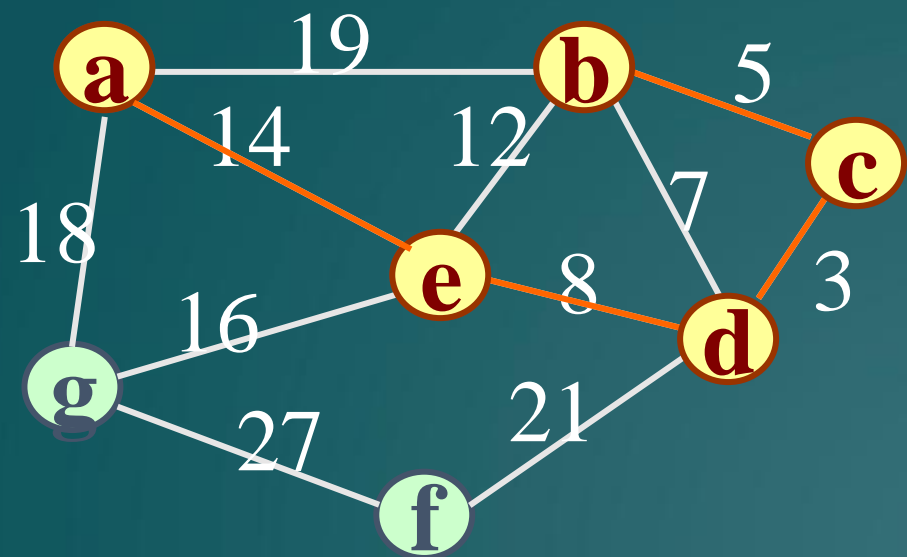
图7.16 普里姆算法构造最小生成树的过程

设置一个辅助数组closedge，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。对每个顶点 $v_i \in V - U$ ，在辅助数组中存在一个相应分量closedge[i-1]，它包括两个域，其中lowcost存储该边上的权。显然：

$$\text{closedge}[i-1].\text{lowcost} = \text{Min}\{\text{cost}(u,v) | u \in U\}$$

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

例如:



<div>closededge</div>	<div>0 a</div>	<div>1 b</div>	<div>2 c</div>	<div>3 d</div>	<div>4 e</div>	<div>5 f</div>	<div>6 g</div>
<div>Adjvex</div>		<div>c</div>	<div>d</div>	<div>e</div>	a	<div>d</div>	<div>e</div>
<div>Lowcost</div>		<div>5</div>	<div>3</div>	<div>8</div>	<div>14</div>	<div>21</div>	<div>16</div>

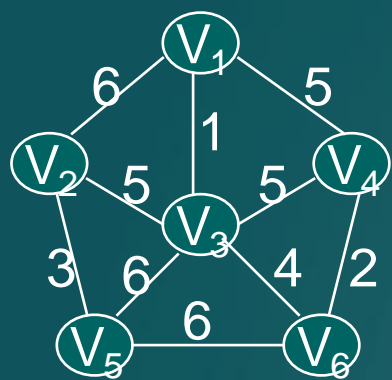
```
void MiniSpanTree_P(MGraph G, VertexType u) {  
    //用普里姆算法从顶点u出发构造网G的最小生成树  
    k = LocateVex ( G, u );  
    for ( j=0; j<G.vexnum; ++j ) // 辅助数组初始化  
        if (j!=k)  
            closedge[j] = { u, G.arcs[k][j].adj };  
    closedge[k].lowcost = 0;    // 初始 , U = {u}  
    for (i=0; i<G.vexnum;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

算法 7.9

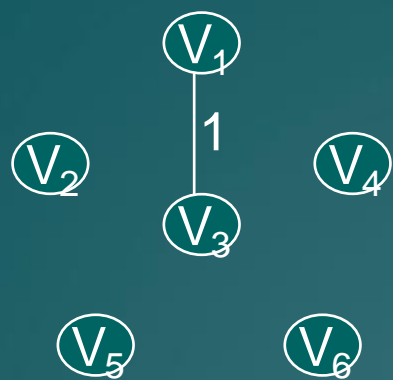
```
k = minimum(closedge);  
    // 求出加入生成树的下一个顶点(k)  
printf(closedge[k].adjvex, G.vexs[k]);  
    // 输出生成树上一条边  
closedge[k].lowcost = 0; // 第k顶点并入U集  
for (j=0; j<G.vexnum; ++j)  
    //修改其它顶点的最小边  
    if (G.arcs[k][j].adj < closedge[j].lowcost)  
        closedge[j] = { G.vexs[k], G.arcs[k][j].adj };
```

克鲁斯卡尔算法的基本思想：

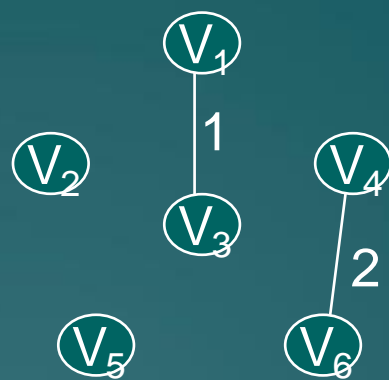
- **考虑问题的出发点:** 为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能的小。
- **具体做法:** 先构造一个只含 n 个顶点的子图 SG ，然后从权值最小的边开始，若它的添加不使 SG 中产生回路，则在 SG 上加上这条边，如此重复，直至加上 $n-1$ 条边为止。



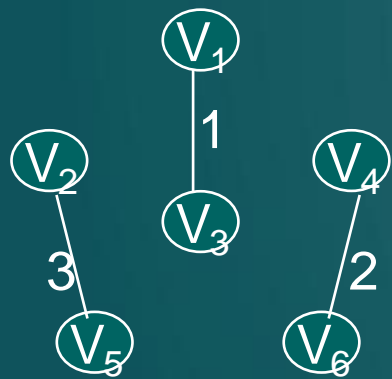
(a)



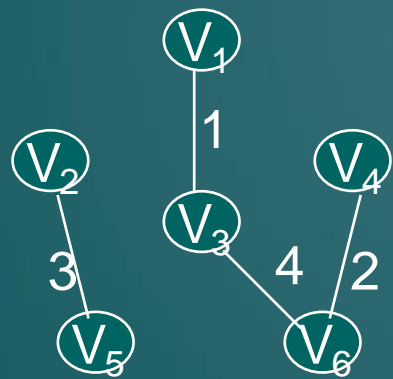
(b)



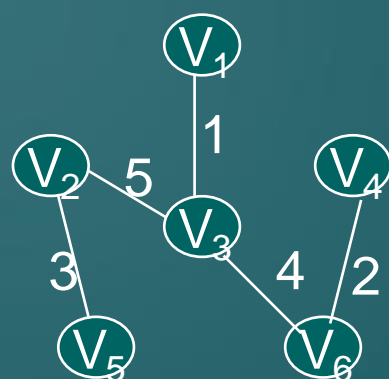
(c)



(d)



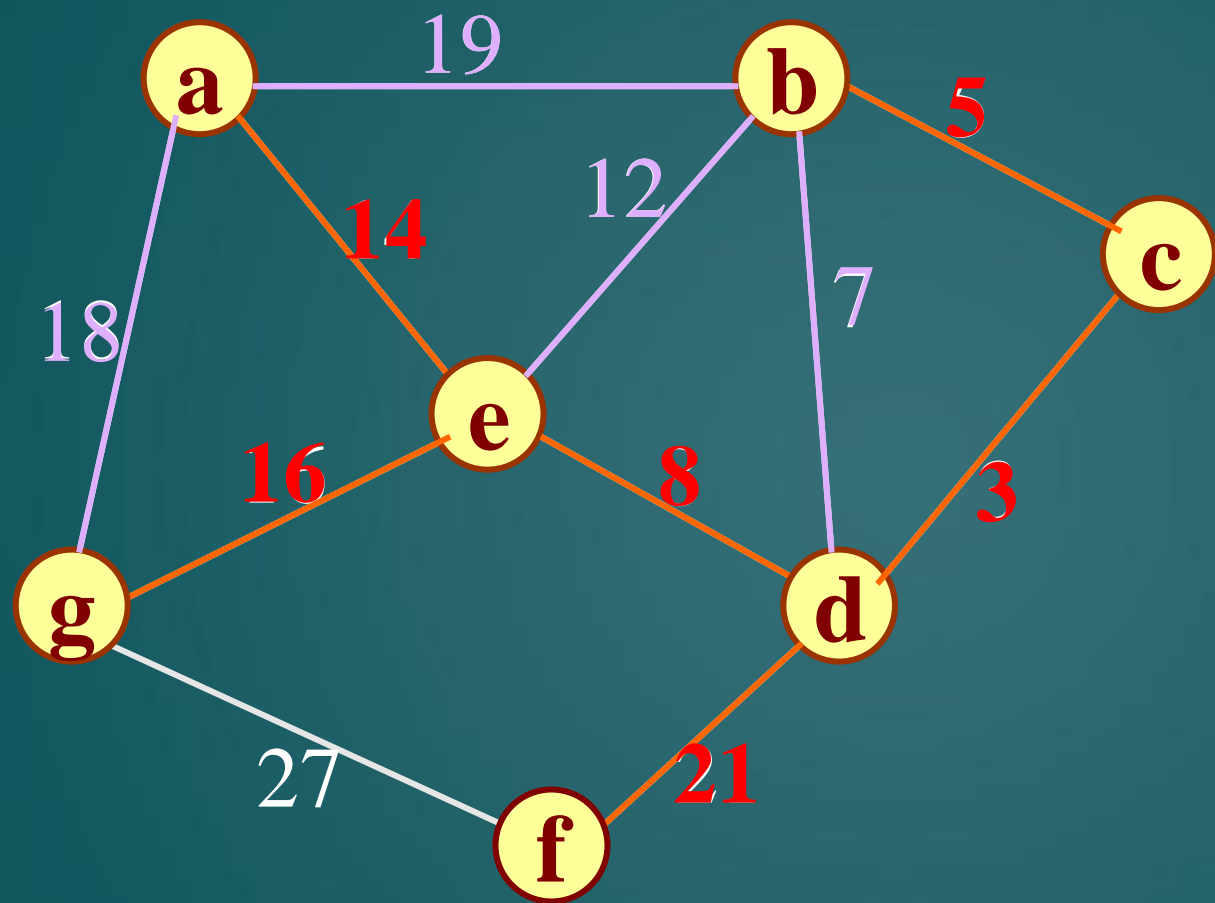
(e)



(f)

图7.18 克鲁斯卡尔算法构造最小生成树的过程

例如:



算法描述:

构造非连通图 $ST=(V,\{\})$;

$k = i = 0$; // k 计选中的边数

while ($k < n-1$) {

$++i$;

 检查边集 E 中第 i 条权值最小的边 (u,v) ; 若 (u,v) 加入 ST

 后不使 ST 中产生回路 ,

 则 输出边 (u,v) ; 且 $k++$;

}

比较两种算法

算法名

普里姆算法

克鲁斯卡尔算法

时间复杂度

$O(n^2)$

$O(e \log e)$

适应范围

稠密图

稀疏图