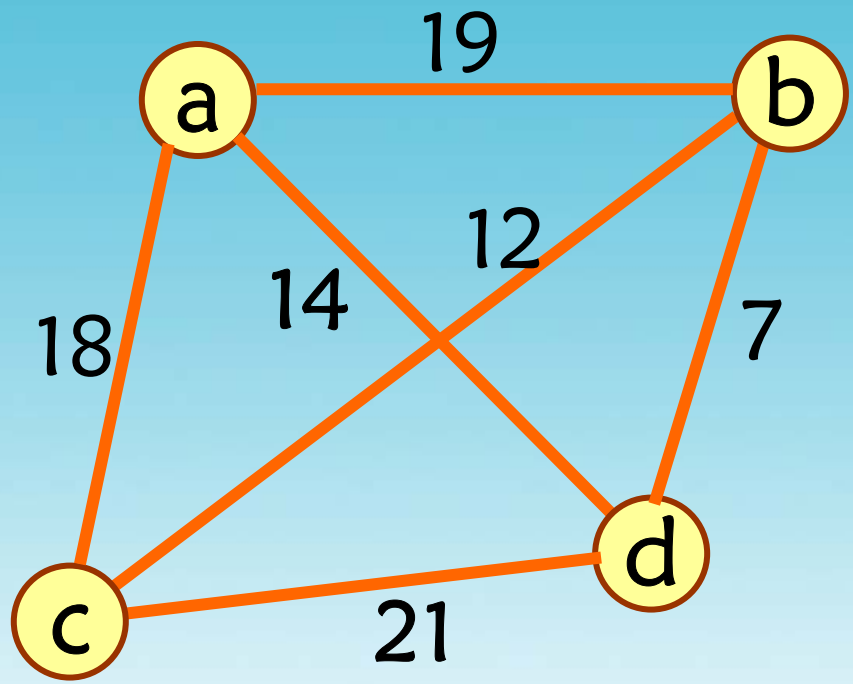
# 8.4 最小生成树

## 问题描述：

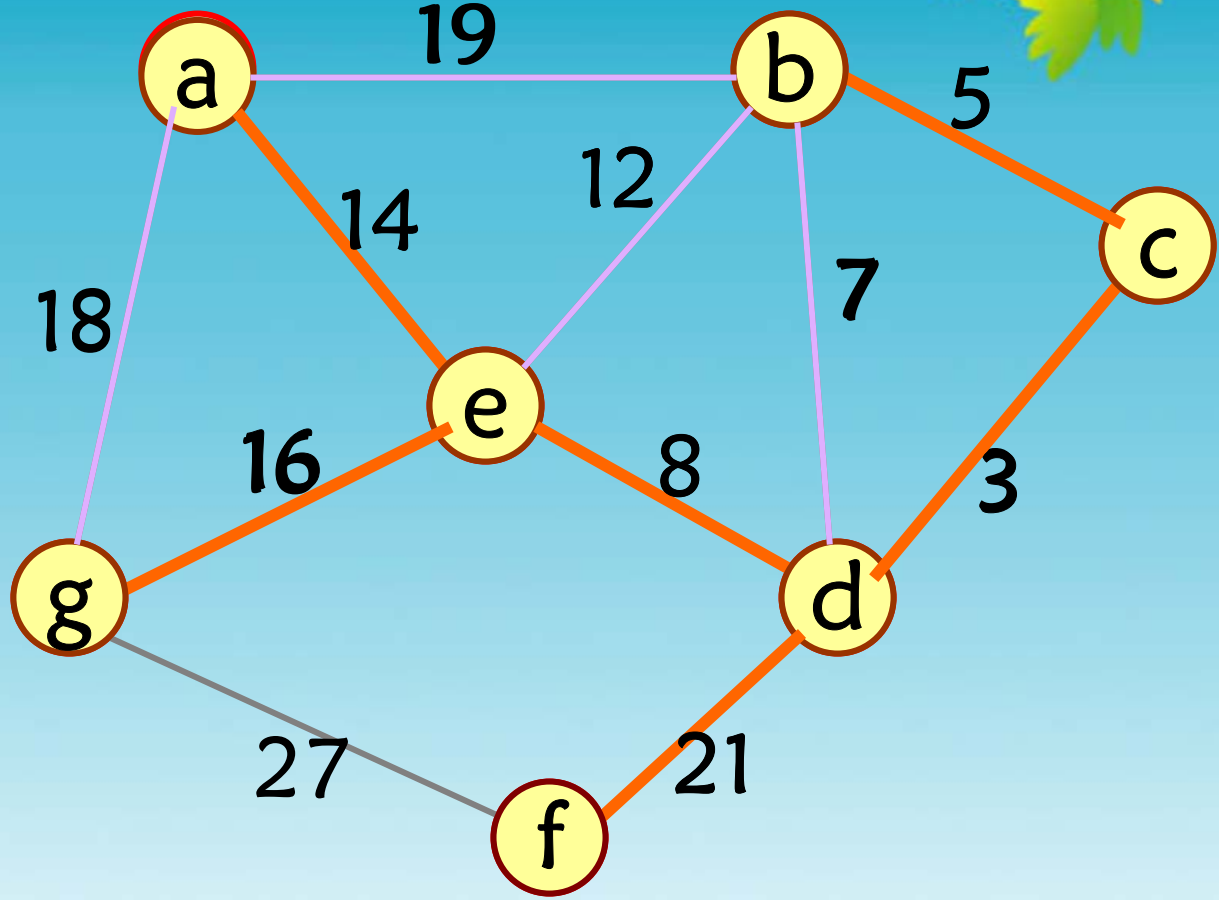
假设要在 n 个城市之间建立通讯联络网，则连通 n 个城市只需要修建 n-1条线路，如何在最节省经费的前提下建立这个通讯网？



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，即最小生成树。

例如：



# 算法描述:

构造非连通图 ST=(V,{});

k = i = 0;  // k 统计已选中的边数

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++;

}

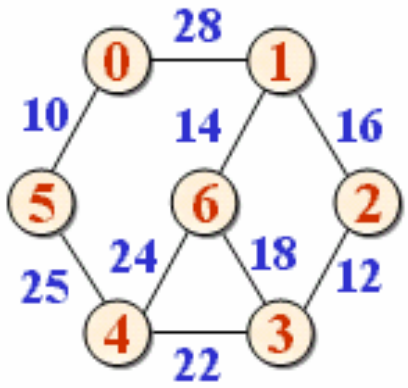
# 堆 （结构如何？）

```
template <class T, class E>
struct MSTEdgeNode    //树边结点的类定义
{
      int tail, head;                  //两顶点位置
      E cost;      // 关系比较<的重载
};

MinHeap <MSTEdgeNode<T, E>> H(m); //最小堆
```

# 图的存储结构

- 邻接矩阵

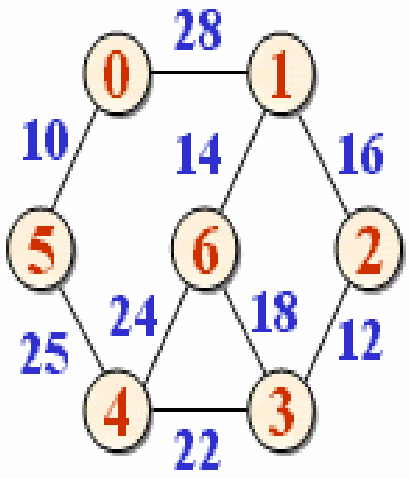



$$\begin{pmatrix} 0 & 28 & \infty & \infty & \infty & 10 & \infty \\ 28 & 0 & 16 & \infty & \infty & \infty & 14 \\ \infty & 16 & 0 & 12 & \infty & \infty & \infty \\ \infty & \infty & 12 & 0 & 22 & \infty & 18 \\ \infty & \infty & \infty & 22 & 0 & 25 & 24 \\ 10 & \infty & \infty & \infty & 25 & 0 & \infty \\ \infty & 14 & \infty & 18 & 24 & \infty & 0 \end{pmatrix}$$

# 初始化操作

图中所有边的数据（结点，结点，权）插入堆中。

```
for (u = 0; u < n; u++)  // 图采用邻接矩阵
  for (v = u+1; v < n; v++)
    if ( Edge[u][v]<maxWeight)
     {  ed.tail = u;  ed.head = v;
        ed.cost = Edge[u][v];
        //插入堆
        H.Insert(ed);
     }
```




$$\begin{pmatrix} 0 & 28 & \infty & \infty & \infty & 10 & \infty \\ 28 & 0 & 16 & \infty & \infty & \infty & 14 \\ \infty & 16 & 0 & 12 & \infty & \infty & \infty \\ \infty & \infty & 12 & 0 & 22 & \infty & 18 \\ \infty & \infty & \infty & 22 & 0 & 25 & 24 \\ 10 & \infty & \infty & \infty & 25 & 0 & \infty \\ \infty & 14 & \infty & 18 & 24 & \infty & 0 \end{pmatrix}$$

```
count = 1;                    //最小生成树边数计数
while (count < n) {           //反复执行，取n-1条边
   H.Remove(ed);              //从堆中取权值最小的边
   //检查该边的两个顶点是否在同一集合中
   //查找出该两顶点所在集合的根u与v——并查集
   u = F.Find(ed.tail);     v = F.Find(ed.head);
   if (u != v)
    {                 //不是同一集合,不连通
      F.Union(u, v);  //集合合并,连通它们——并查集
      MST.Insert(ed); //将该边放入生成树MST中
      // cout <<ed.tail<<ed.head<<ed.cost  输出边
      count++;
    }
  }
};
```

# 普里姆(prim)算法

算法步骤：

1.设定其中一个结点为出发点；

2.分组：出发点为第一组，其余结点为第二组。

3.在一端属于第一组和另一端属于第二组的边中选择一条权值最小的一条。

4.把原属于第二组的结点放入第一组中。

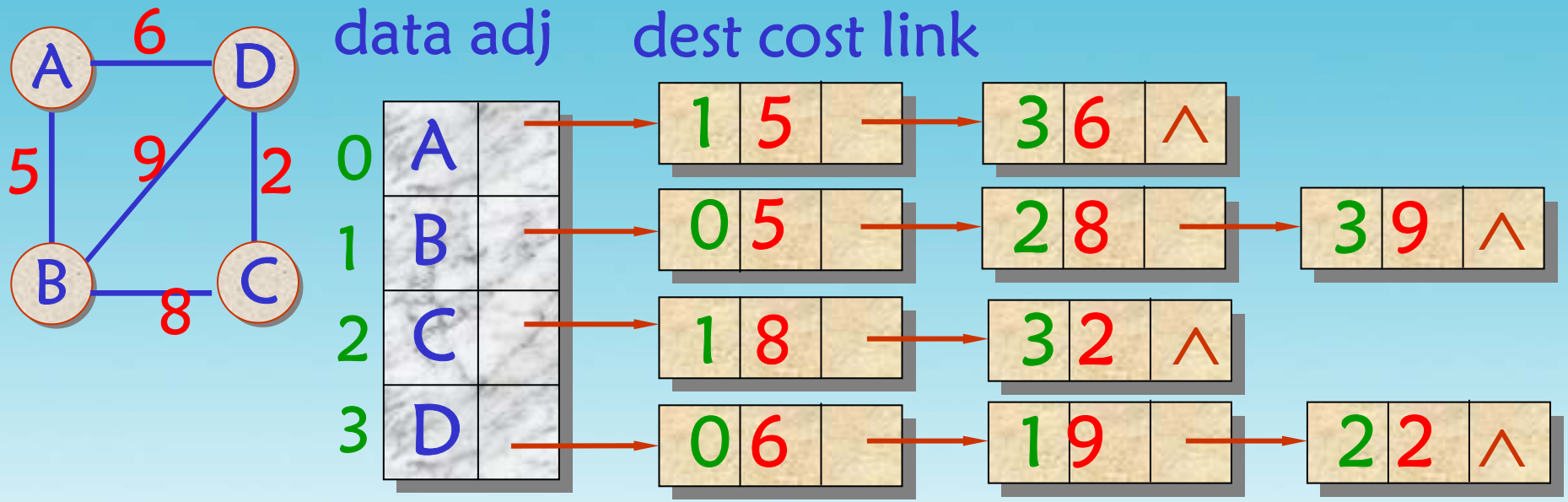
5.反复2，3两步，直到第二组为空为止。

华南师范大学计算机学院 讲稿

# 算法设计

## （1）图的存储结构

### 邻接表



## （2）图的存储结构

邻接矩阵

$$\begin{pmatrix} 0 & 28 & \infty & \infty & \infty & 10 & \infty \\ 28 & 0 & 16 & \infty & \infty & \infty & 14 \\ \infty & 16 & 0 & 12 & \infty & \infty & \infty \\ \infty & \infty & 12 & 0 & 22 & \infty & 18 \\ \infty & \infty & \infty & 22 & 0 & 25 & 24 \\ 10 & \infty & \infty & \infty & 25 & 0 & \infty \\ \infty & 14 & \infty & 18 & 24 & \infty & 0 \end{pmatrix}$$

# 算法设计

```
template <class T, class E>
struct MSTEdgeNode    //树边结点的类定义
{
      int  head;   //未加入生成树的结点编号
      int tail; //已加入生成树的结点编号
      E cost;      // 关系比较<的重载
};

MinHeap <MSTEdgeNode<T, E>> H(m); //最小堆
```

# 算法的描述

```
Edge[u][u] = true;        //u 加入生成树
count = 1;  // 记录已加入生成树的边数
while (count<n)
{   for (v=0;v<n;i++) // 逐个查看v是否在生成树中
   {  if ((Edge[v][v]==0) &&Edge[u][v]<maxWeight)
      {  ed.tail=u;    ed.head =v ;
         ed.cost = Edge[u][v];
         //(u,v,w)加入堆
         H.Insert(ed);
      }
 }
```

$$\begin{pmatrix} 0 & 28 & \infty & \infty & \infty & 10 & \infty \\ 28 & 0 & 16 & \infty & \infty & \infty & 14 \\ \infty & 16 & 0 & 12 & \infty & \infty & \infty \\ \infty & \infty & 12 & 0 & 22 & \infty & 18 \\ \infty & \infty & \infty & 22 & 0 & 25 & 24 \\ 10 & \infty & \infty & \infty & 25 & 0 & \infty \\ \infty & 14 & \infty & 18 & 24 & \infty & 0 \end{pmatrix}$$

```
while (!H.IsEmpty() && count < n) {
  H.Remove(ed);        //从堆中删除最小权的边
  if (Edge[ed.head] [ed.head]==0)//是否符合选择要求
   {
     MST.Insert(ed); //将该边加入最小生成树
     // cout <<ed.tail<<ed.head<<ed.cost 输出边
     u = ed.head;
     Edge[u][u] = true; //u加入生成树集合中
     count++;
     break;
   }
  }
}
```