数据结构

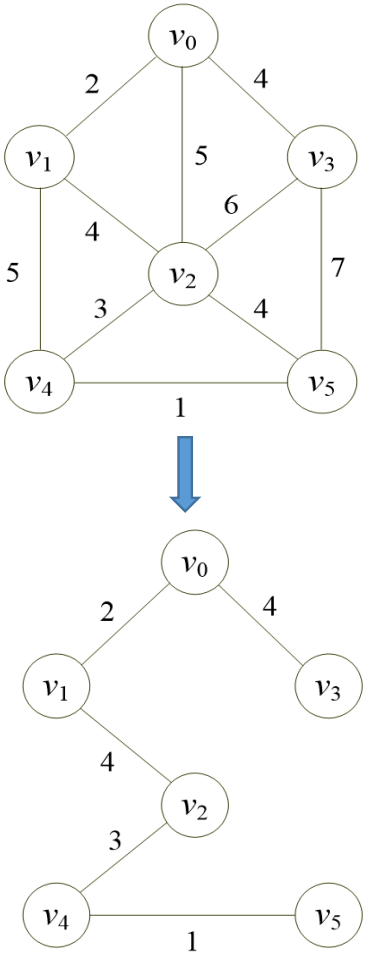
8.4 图的最小生成树算法

Outline

- 最小生成树的概念与应用
- 贪心算法设计的基本架构
- Prim算法的设计
- Kruskal算法的设计
- 两种算法的比较

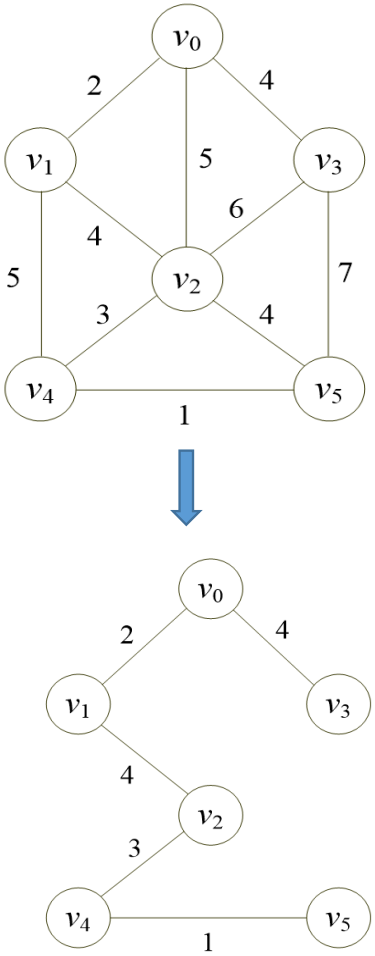
无向网的最小生成树的概念

- 生成树的概念
  - ✓ n个顶点的连通网络的生成树有n个顶点、n-1条边。
  - ✓ 生成树是连通图的**极小连通子图**。
- 什么是最小生成树？
  - ✓ 生成树各边的权值总和称为**生成树的权**。
  - ✓ 权最小的生成树称为最小生成树。



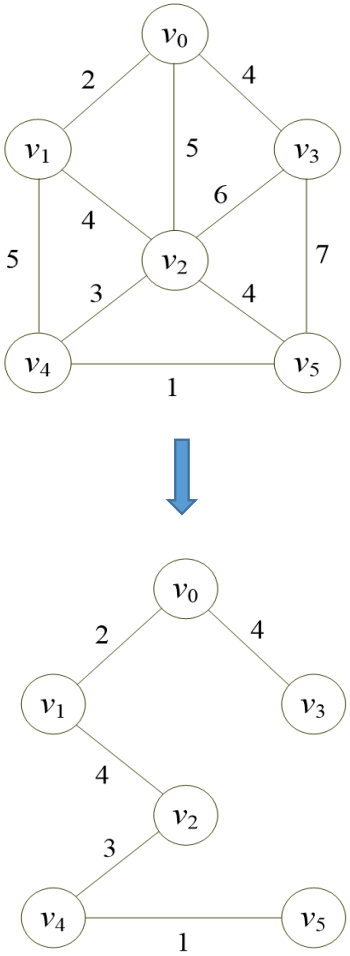
无向网的最小生成树问题的应用

- **最小生成树的应用实例**
  - ✓ **假设要在n个城市之间建立通信网。令图G的顶点表示城市，边表示连接两个城市的通信线路，边的权值表示通信线路的长度或代价。在n个城市间构造通信网最少需要n-1条线路。**
  - ✓ **问：如何选择这n-1条边，使得构造这个通信网总的代价最小?**



最小生成树的构造方法

- **最小生成树的构造目标是从给定连通网的边集E中选择一个含有n-1条边的最优子集。**
- **采用蛮力法求解的复杂度可达到指数级。**
- **可采用基于贪心策略的高效求解算法。**



基于贪心策略设计算法的基本步骤

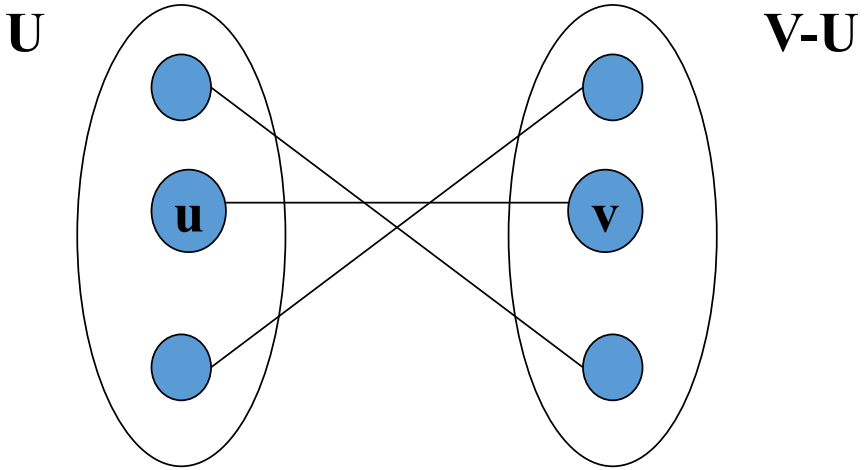
- 简单地说，贪心算法设计的基本方法是：
  - ✓ 将问题的求解过程转换为执行**一系列贪心选择**的过程；
  - ✓ 执行每一步贪心选择时，只需要根据当前的状态依据一个设定好的**贪心准则**做一个**局部最优**选择。

贪心准则: MST性质

- MST性质

设 G=(V,E)是一个无向连通网，给定图G的一个割（U,V-U），U是顶点集V的一个真子集；

若(*u*,*v*)是G中所有的一个顶点在U，另一个顶点不在U的边（交叉边）中具有最小权值的一条边，则一定存在G的一棵最小生成树包括此边。



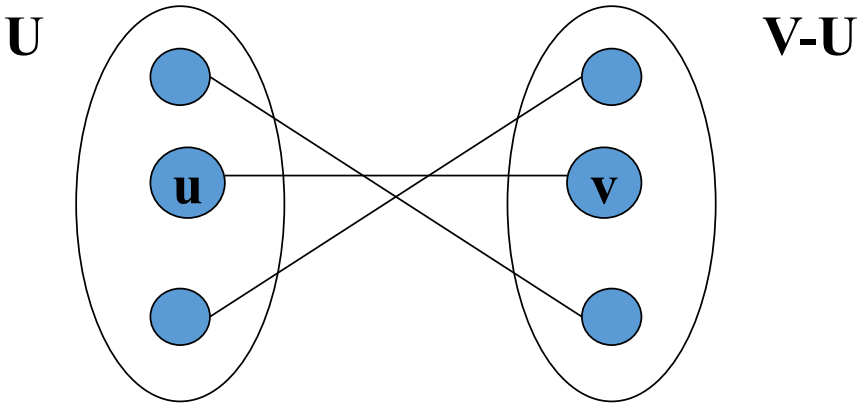
贪心准则: MST性质

- MST性质的证明

反证法：

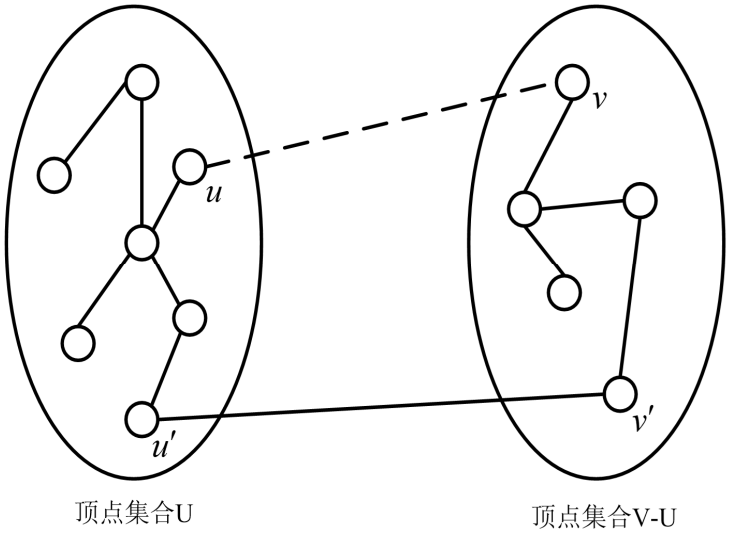
首先假设图G中任何一棵最小生成树都不包含最小交叉边(u,v)；

然后，据此推出矛盾，证明假设不成立，从而证明MST性质的正确性。



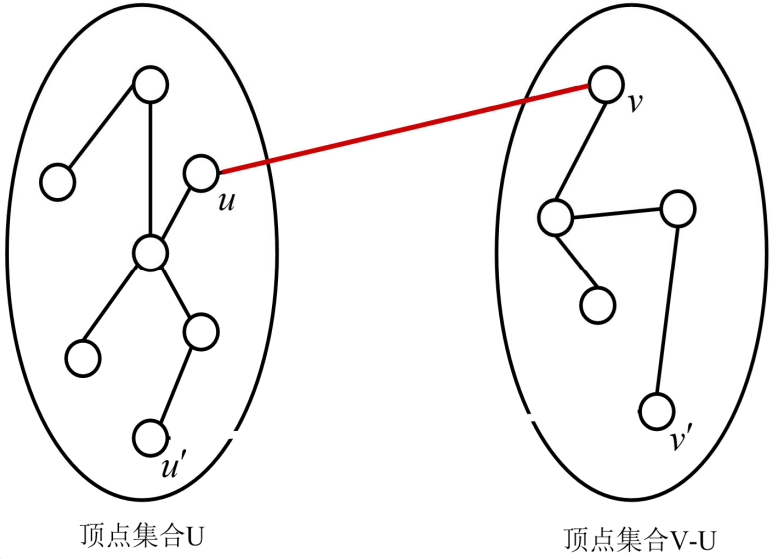
MST性质的证明

- MST性质的证明



最小生成树 T
(不包含最小边(*u*,*v*))

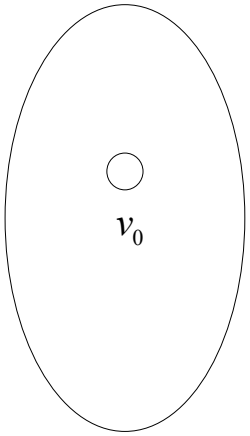
推出矛盾，
假设不成立！



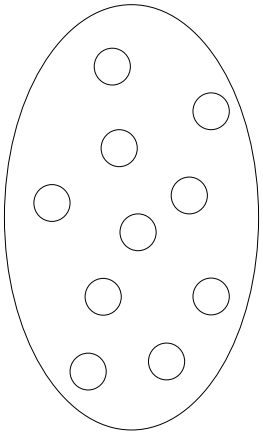
生成树 T'
(包含最小边(*u*,*v*))

Prim 算法的基本思想

- **基于贪心策略的最小生成树T的构造过程**
  - ✓ **初始状态：**
    - ➢ **生成树T中仅含有一个构造的初始顶点，边集为空**
  - ✓ **依据MST贪心准则，循环迭代执行n-1次：**
    - ➢ **在当前图割的状态下，选择一条最小交叉边；**
    - ➢ **将这条交叉边及其关联的另一个顶点加入生成树T中。**



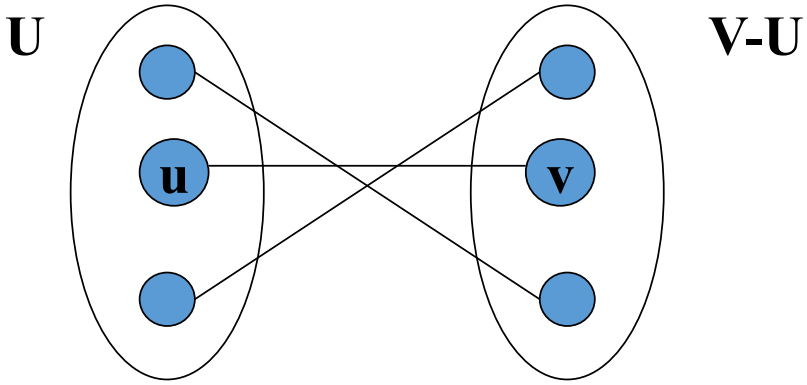
顶点集合U



顶点集合V-U

实现Prim 算法的关键问题

- 每次如何从图的割中选择一条权值最小的交叉边(u,v)?
  - ✓ 蛮力法效率低!
  - ✓ 注意增加一个个顶点到最小生成树中的过程实际上是一种**增量式**的变化
  - ✓ 我们感兴趣的是从**生成树T外**的每个顶点到**生成树T中**各顶点的最小权值。

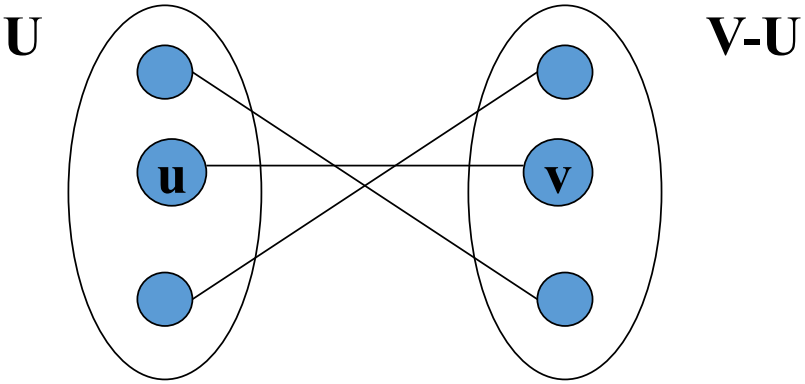


实现Prim 算法的关键问题

- **为实现上述思想，需要引入一个辅助数组 miniedges[] 记录下列信息：**
  - **从生成树T外的每个顶点（即集合V-U中的每个顶点）到生成树T中顶点（即集合U中顶点）的最小交叉边；**
  - **这些最小交叉边相应的权值。**

```
template<class T>
struct Edge{
        T  adjvex;
        float  lowcost;
};
```

$miniedges[i].lowcost = Min\{cost(v_i,u) | u \in U\}$



Prim 算法的框架描述

1. 从连通网 $N=\{V, E\}$ 中的选择任一顶点v_0加入到生成树 T 的顶点集合 U中，并为集合$V-U$中的各顶点置最小交叉边.

2. **while (生成树T中顶点数目 < n) {**

   从集合$V-U$中各顶点对应的最小交叉边中选取最小权值边(u, v);

   将边(u, v) 及其在集合$V-U$中的顶点 v，加到生成树 T 中；

   调整集合$V-U$中各顶点对应的最小权值；

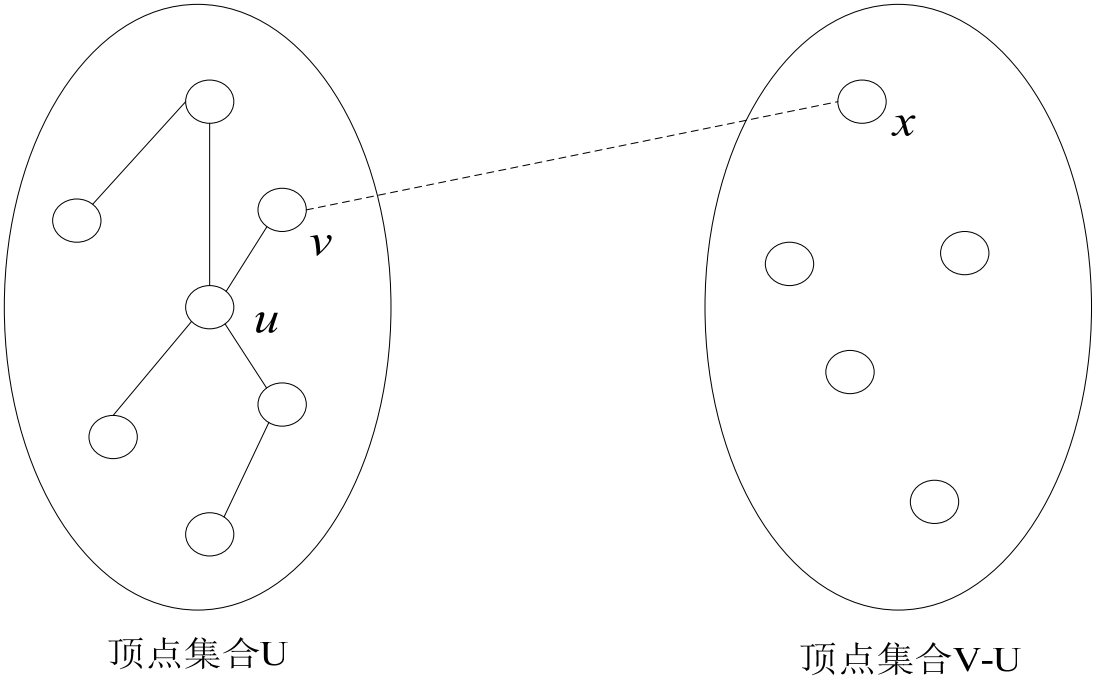
}

V-U中各顶点对应最小权值的调整方法

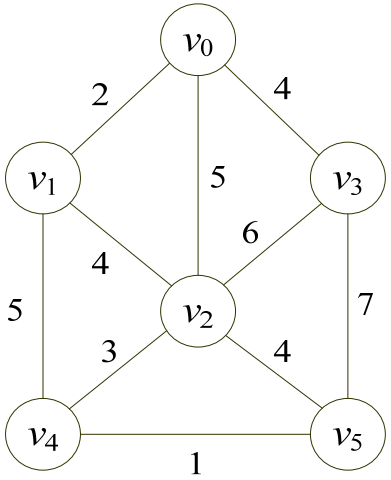
从集合$V-U$中各顶点对应的最小交叉边中选取最小权值边(u, v);

将边(u, v) 及其在集合$V-U$中的顶点 v，加到生成树 T 中；

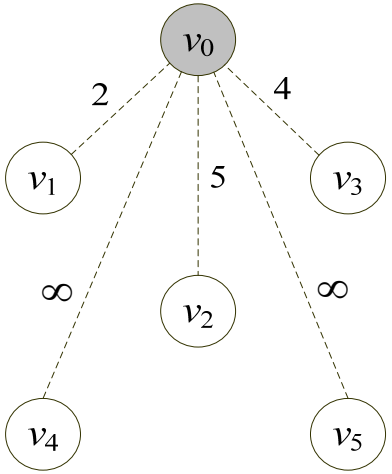
调整集合$V-U$中各顶点对应的最小权值；



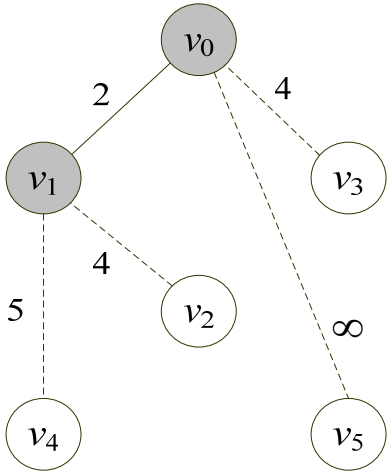
Prim算法构造最小生成树的过程示例



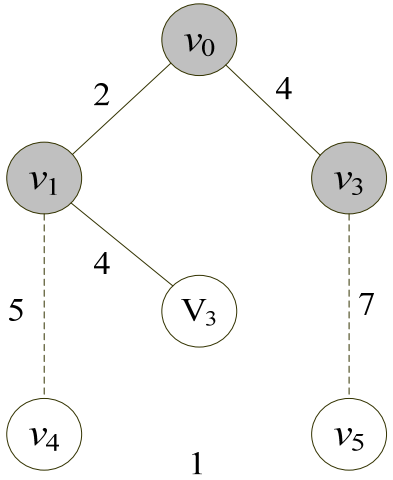
(a) 无向连通网N_2



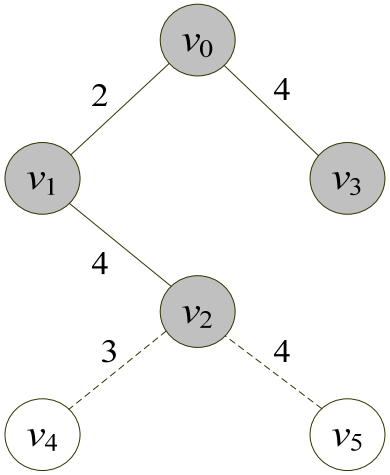
(b)



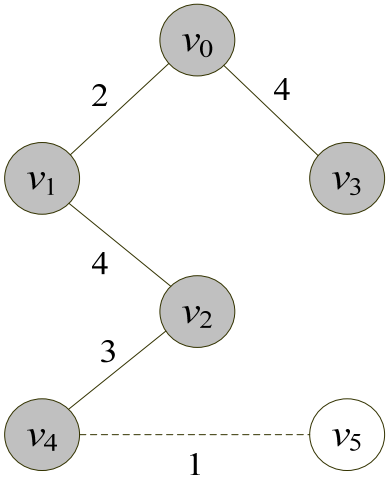
(c)



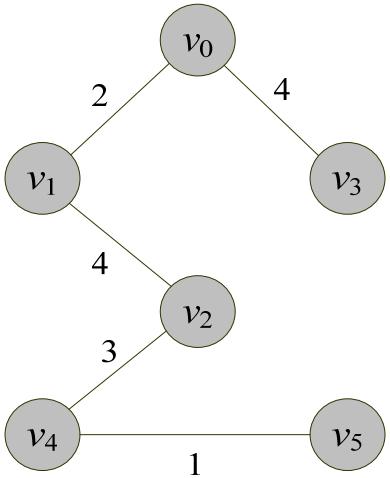
(d)



(e)



(f)



(g)

```
template<class T>
void MGraph<T>::Prim(int v) {
        Edge<char>* miniedges= new Edge<T>[vexnum];
        for(i=0;i<vexnum;i++){
            miniedges[i].adjvex=GetVexValue(v);
            miniedges[i].lowcost=GetEdgeValue(v, i);
        }
        miniedges[v].lowcost=0;
        for(i=1;i<vexnum;i++){
            k=MiniNum(miniedges);
            cout<<miniedges[k].adjvex<<"-->"<<GetVexValue(k)<<endl;
            miniedges[k].lowcost=0;
            for(j=0;j<vexnum;j++){
                if(GetEdgeValue(k,j)<miniedges[j].lowcost){
                    miniedges[j].adjvex=GetVexValue(k);
                    miniedges[j].lowcost=GetEdgeValue(k,j);
                }
            }
        }
        delete []miniedges;
}
```

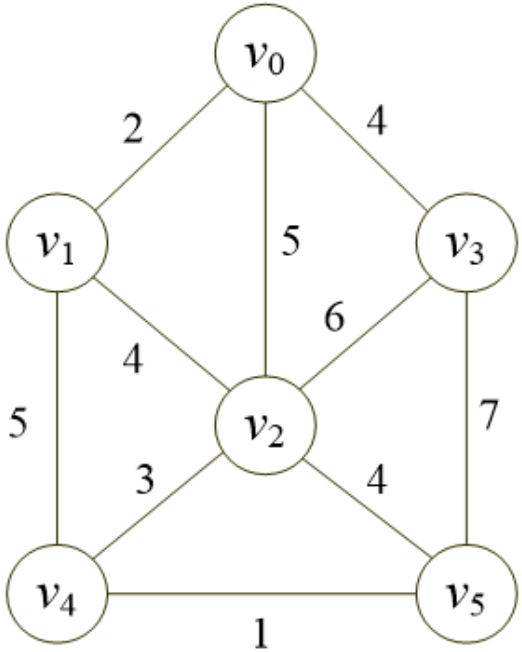
```
template<class T>
struct Edge{
        T  adjvex;
        float  lowcost;
 };
```

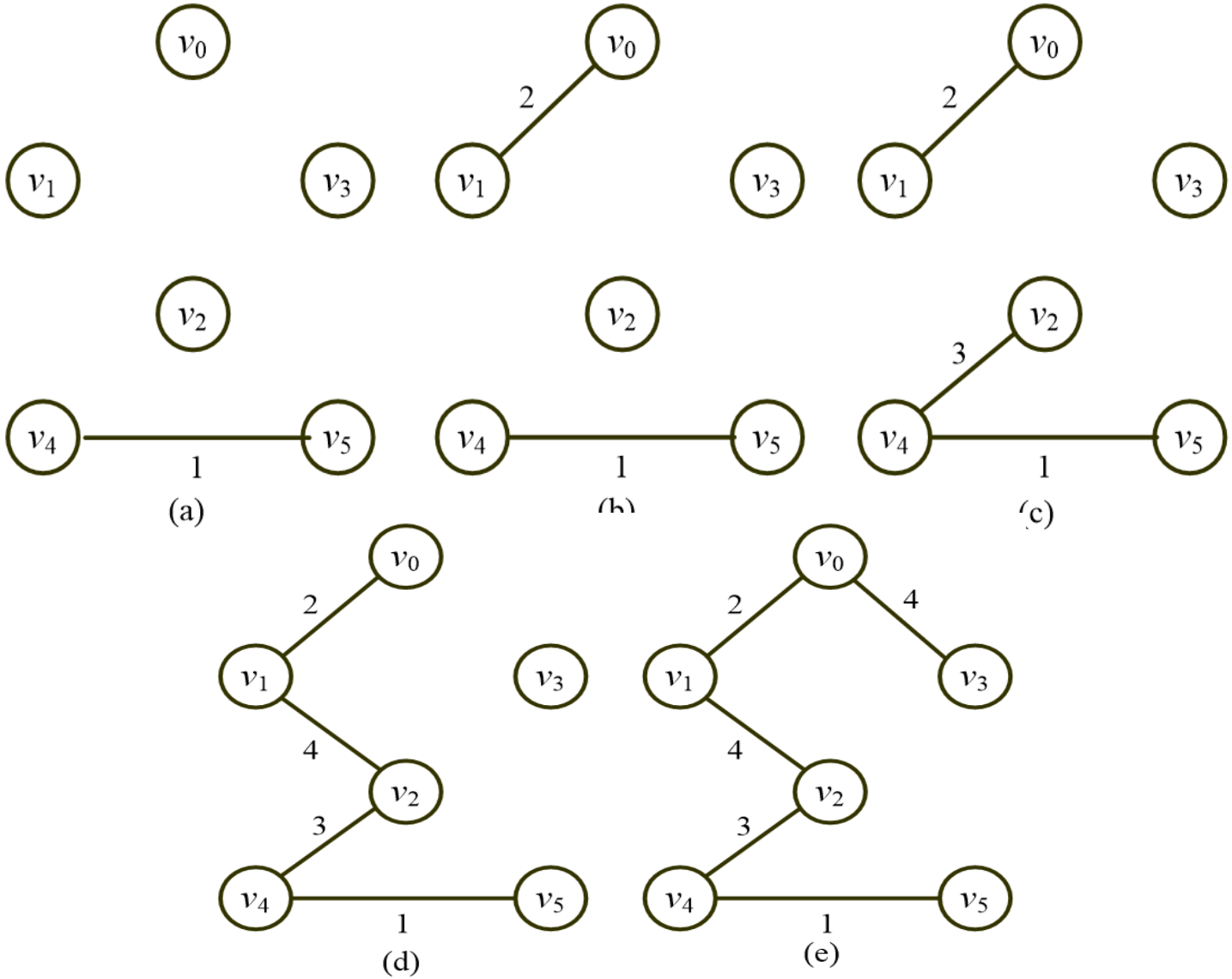
算法的时间复杂度为$O(n^2)$

Kruskal 算法的基本思想

- 基于贪心策略的最小生成树T的构造过程
  - ✓ 初始状态：
    - ➢ 生成树T中包含有n个顶点，边集为空，树中每个顶点自成一个连通分量。
  - ✓ 依据**MST贪心准则**，循环执行直到所有顶点在同一个连通分量上为止：
    - ➢ 在连通网的剩余边集中选出一条具有最小权值的边,若该边的两个顶点落在不同的连通分量上，则将此边加入到 T 中；
    - ➢ 否则将此边舍去，重新选择一条权值最小的边。

Kruskal 算法构造最小生成树的过程示例





Kruskal 算法的框架描述

```
T = (V, ∅ );
while  ( T中所含边数 < n-1 ){
      从E中选取当前最小权值边 (u,v);
      从E中删除边(u,v);
      if  ( (u,v) 并入T之后不产生回路 )
          将边 (u,v) 并入T中;
}
```

怎么有效地实现？

```
template <class T>
void MGraph<T>::Kruskal(vector<EdgeType> &tree) {
    GetGraph(graph);     //将边的集合按代价cost从小到大放入graph数组
    tree.resize(vexnum-1);
    int *components =new int[vexnum];
    for( i=0;i<vexnum;i++) components[i]=i;
    k=0; j=0;
    while(k<vexnum-1) {    //最小生成树的边有vexnum-1条
        h1=graph[j].head; t1=graph[j].tail;
        h2=components[h1]; t2=components[t1];
        if(h2!=t2){
            tree[k].head=h1; tree[k].tail=t1;
            tree[k].cost=graph[j].cost;
            k++;
            for(i=0;i<vexnum;i++)                    {
                if(components[i]==t2) components[i]=h2;
        }
        j++;
     }
     delete []components;
}
```

```
template<class T>
struct EdgeType {
    T head, tail;
     int cost;
};
```

```
template <class T>
void MGraph<T>::Kruskal(vector<EdgeType> &tree) {
    vector<EdgeType> graph;
    getGraph(graph)     //将边集按权值cost从小到大放入graph数组
    MFset<T> set(vexs);    // 初始化并查集
    k = 0; j=0;
    while( k<vexnum - 1 ){
        h1=graph[j].head;  t1=graph[j].tail;
        h2=set.Find(h1);     t2=set.Find(t1);
        if(h2 != t2){
            tree[k].head=h1; tree[k].tail=t1;
            tree[k].cost=graph[j].cost;
            set.Merge(h2, t2);
            k++;
        }
        j++;
    }
}
```

```
template<class T>
struct EdgeType {
    T head, tail;
     int cost;
};
```

算法的时间复杂度如何？

- ✓ 算法的效率与所选择的排序算法的效率以及并查集数据结构的实现效率有关
- ✓ 较好情况下的算法时间复杂度为 $O(e\log_2 e)$ ，e为边数。

两种算法的比较

- **共同点**
  - 都是基于贪心策略的算法
  - 基于相同贪心准则MST性质
- **不同点**
  - Prim算法的特点是生成树中的边总是形成单棵树，而在Kruskal算法中生成树中的边是一个不断生长的森林。
  - 相比较Prim算法而言，Kruskal算法更适用于求解稀疏网的最小生成树。

Thank you for your attention!