7.4 生成树和最小生成树

7.4.1 生成树（spanning tree）

7.4.2 最小生成树（Minimun spanning tree ）

7.4 生成树和最小生成树

7.4.1 生成树（spanning tree）

7.4.2 最小生成树（Minimun spanning tree ）

7.4.1 生成树(spanning tree)

T是G 的生成树当且仅当T 满足如下条件

- T是G 的连通子图
- T包含G 的所有顶点
- T中无回路

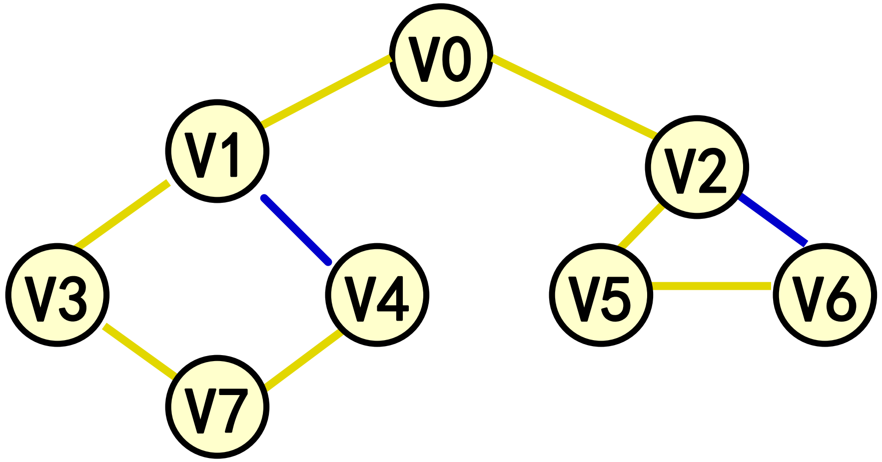
- 生成树是连通图的极小连通子图。所谓极小是指：若在树中任意增加一条边，则将出现一个回路；若去掉一条边，将会使之变成非连通图。
- 利用深（广）度优先搜索可以实现 求深（广）度优先生成树。

7.4.1 生成树(spanning tree)

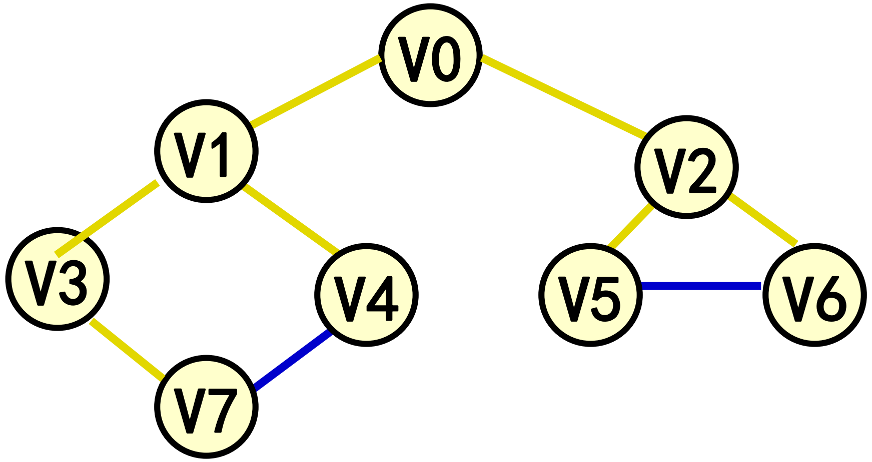
- **无向图的生成树**

  生成树可由遍历过程中所经过的边组成

  - 深度优先生成树
  - 广度优先生成树



深度优先生成树



广度优先生成树

7.4 生成树和最小生成树

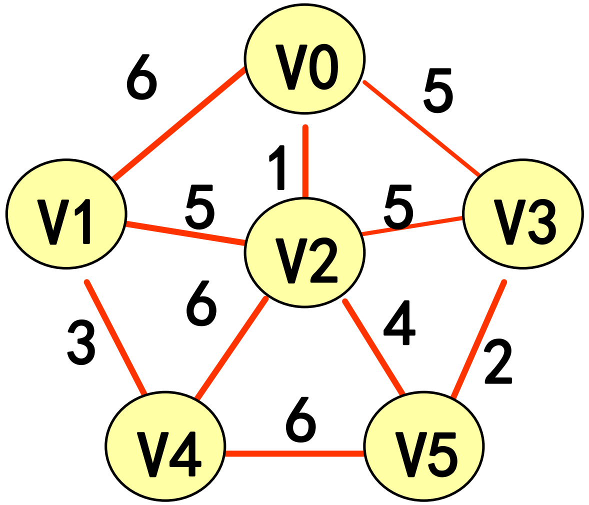
7.4.1 生成树（spanning tree）

7.4.2 最小生成树（Minimun spanning tree ）

7.4.2 最小生成树（Minimun spanning tree ）

例　　要在 n 个城市间建立交通网，要考虑的问题如何在保证 n 点连通的前题下最节省经费?

求解:　连通6个城市且代价最小的交通线路?



- **最小生成树:**

生成树的权(代价）最小的生成树称为最小生成树

7.4.2 最小生成树（Minimun spanning tree ）

该问题等价于：

◆构造最小生成树的准则

- 必须使用且仅使用该连通图中的n-1 条边连接结图中的 n 个顶点;
- 不能使用产生回路的边;
- 各边上的权值的总和达到最小。

算法一： Prim （普里姆算法）

算法二： Kruskal （克鲁斯卡尔算法）

7.4.2 最小生成树（Minimun spanning tree ）

普里姆算法的基本思想:

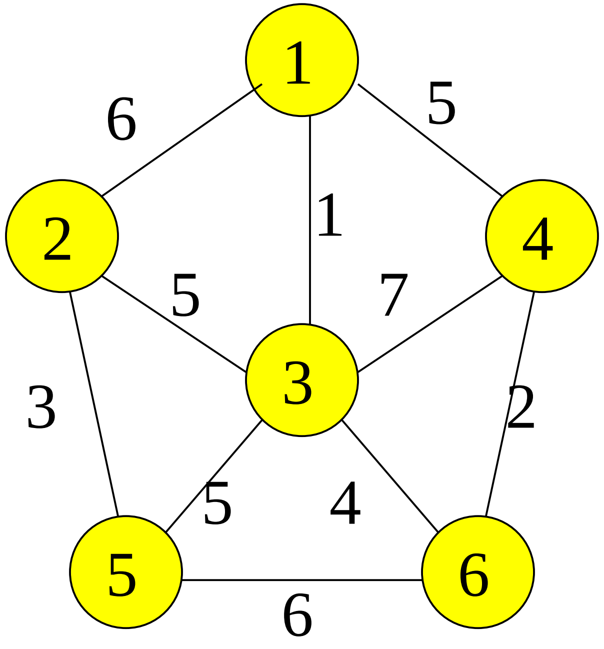
1. 取图中任意一个顶点 v 作为生成树的根，之后往生成树上添加新的顶点 w。

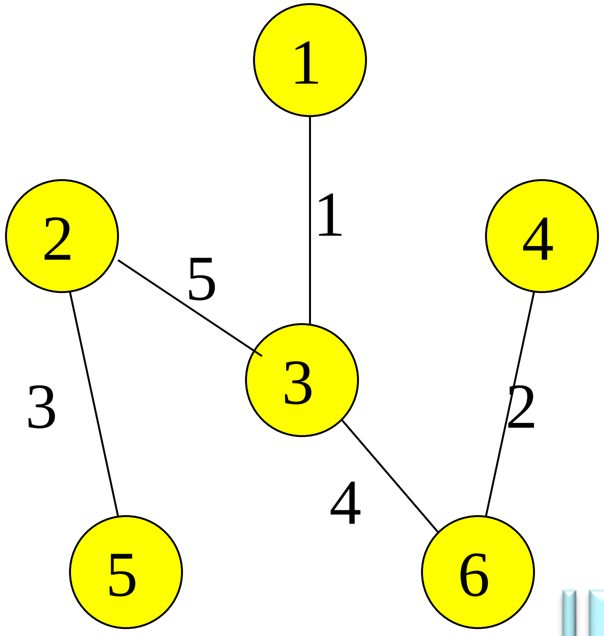
2.在添加的顶点 w 和已经在生成树上的顶点v 之间必定存在一条边，并且该边的权值在所有连通顶点 v 和 w 之间的边中取值最小。

3.继续往生成树上添加顶点，直至生成树上含有 n 个顶点为止。

7.4.2 最小生成树（Minimun spanning tree ）

用Prim算法构造最小生成树





图示

7.4.2 最小生成树（Minimun spanning tree ）

Prim算法基本步骤

设G=（V，E）为一个具有n个顶点的带权的连通网络，T=（U，TE）为构造的生成树。

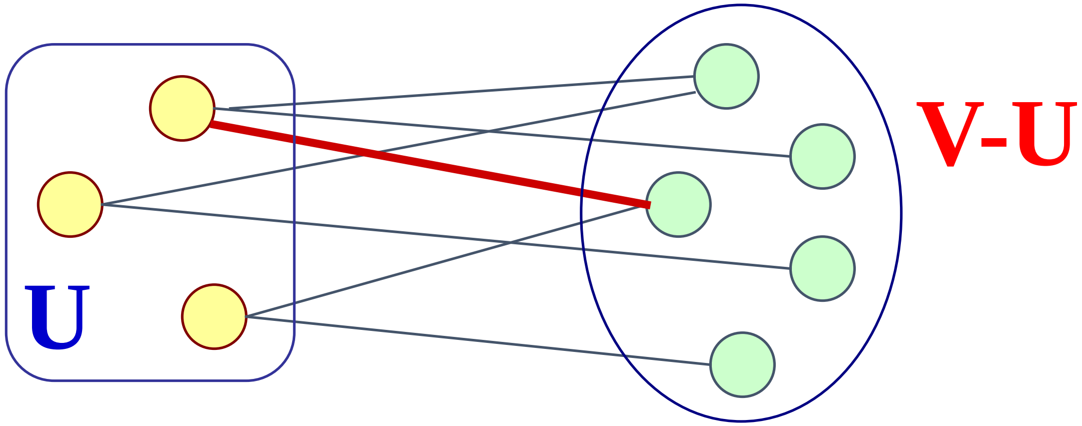
（1）初始时，U={V0}，TE=ϕ；

（2）在所有u∈U 、v∈V-U 的边(u,v)中选择一条权值最小的边，不妨设为（u,v)；

（3）(u,v) 加入TE，同时将v 加入U；

（4）重复(2)、(3)，直到U=V为止；

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。



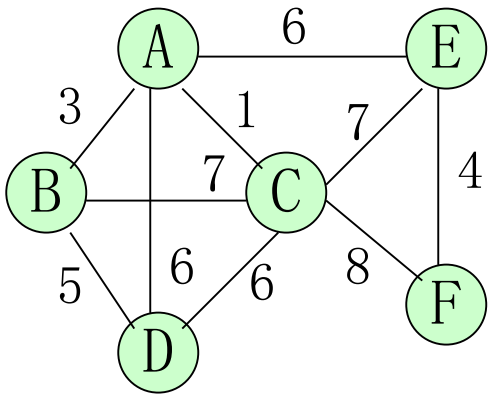
7.4.2 最小生成树（Minimun spanning tree ）

- **Prim算法**

【算法要点】

Void Prim（G，T）
{T= ϕ ;//T 存放加入的边;
U={V0};// U存放加入的结点;
while(V-U!= ϕ)
{设（u,v）是u $\in$ U与v $\in$（V-U）且权最小的边;
T=T $\bigcup$ {（u，v）};
U=U $\bigcup$ {v}
}
}

Prim算法思想：



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

初始化

黄色表示U的顶点，其他为V-U的顶点

A

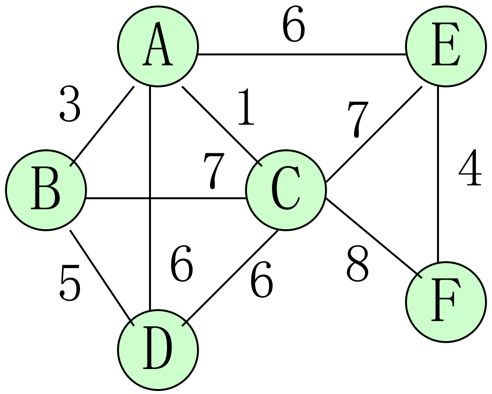
V-U中各顶点到U的最短直接路径LOWCOST[i]：

A	B	C	D	E	F
0	3	1	6	6	∞

相邻顶点CLOSEST[i]：

A	B	C	D	E	F
A	A	A	A	A	A

Prim算法：

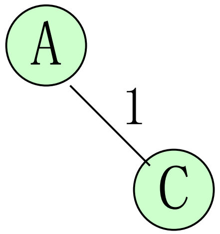


图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

比较大小

0	3	1	6	6	∞



V-U中各顶点到U的最短直接路径LOWCOST[i]：

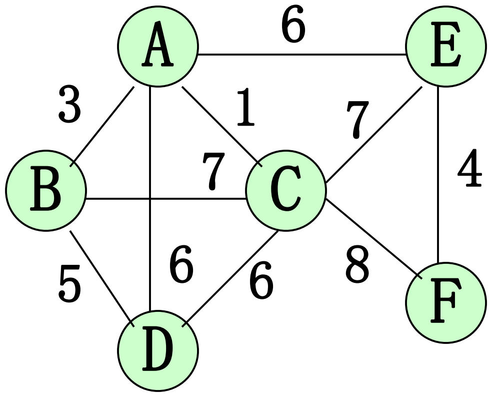
A	B	C	D	E	F
0	3	1	6	6	8

A	A	A	A	A	A

相邻顶点CLOSEST[i]：

A	A	A	A	A	C

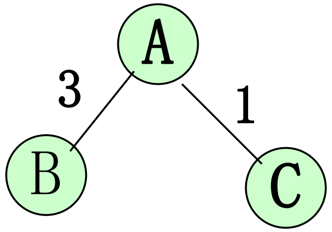
Prim算法：



图G

$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E \\ F
\end{array}
\begin{array}{c}
\begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\left(\begin{array}{cccccc}
0 & 3 & 1 & 6 & 6 & \infty \\
3 & 0 & 7 & 5 & \infty & \infty \\
1 & 7 & 0 & 6 & 7 & 8 \\
6 & 5 & 6 & 0 & \infty & \infty \\
6 & \infty & 7 & \infty & 0 & 4 \\
\infty & \infty & 8 & \infty & 4 & 0
\end{array}\right)
\end{array}
$$

比较大小



V-U中各顶点到U的最短直接路径LOWCOST[i]：

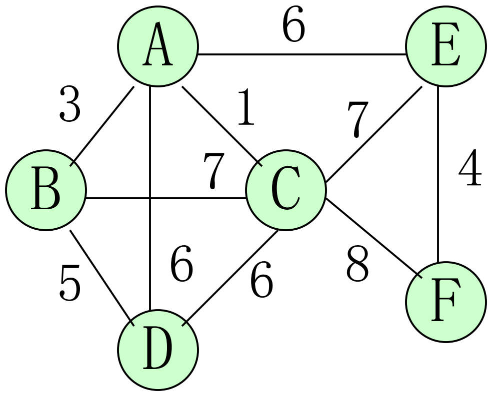
0	3	1	6	6	8
0	3	1	5	6	8

A	B	C	D	E	F
A	A	A	A	A	C

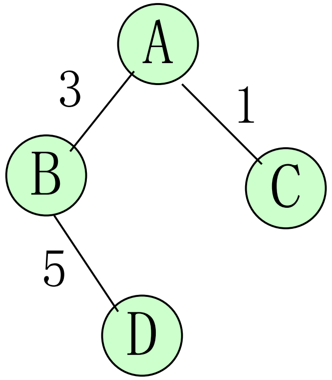
相邻顶点CLOSEST[i]：

A	A	A	B	A	C

Prime算法:



图G



$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E \\ F
\end{array}
\begin{array}{c}
\begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\begin{pmatrix}
0 & 3 & 1 & 6 & 6 & \infty \\
3 & 0 & 7 & 5 & \infty & \infty \\
1 & 7 & 0 & 6 & 7 & 8 \\
6 & 5 & 6 & 0 & \infty & \infty \\
6 & \infty & 7 & \infty & 0 & 4 \\
\infty & \infty & 8 & \infty & 4 & 0
\end{pmatrix}
\end{array}
$$

比较大小

V-U中各顶点到U的最短直接路径LOWCOST[i]:

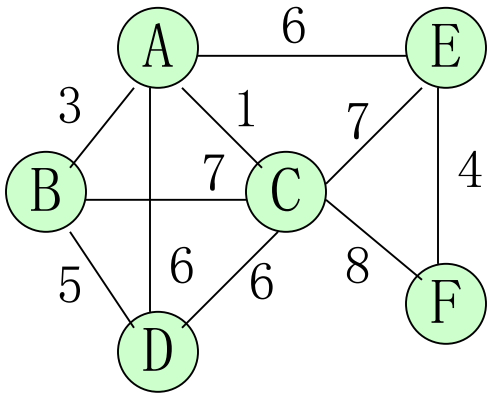
0	**3**	**1**	**5**	**6**	**8**
0	**3**	**1**	**5**	**6**	**8**

A	B	C	D	E	F
A	**A**	**A**	**B**	**A**	**C**

相邻顶点CLOSEST[i]:

A	**A**	**A**	**B**	**A**	**C**

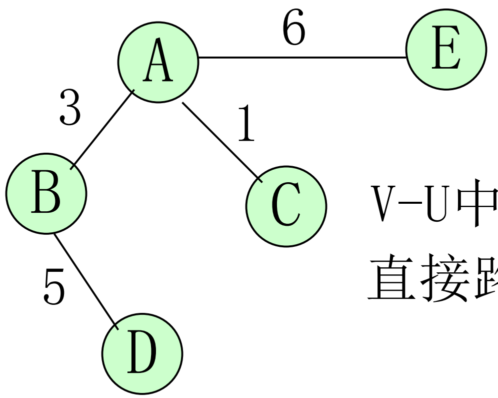
Prime算法：



图G

$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 3 & 1 & 6 & 6 & \infty \\
B & 3 & 0 & 7 & 5 & \infty & \infty \\
C & 1 & 7 & 0 & 6 & 7 & 8 \\
D & 6 & 5 & 6 & 0 & \infty & \infty \\
E & 6 & \infty & 7 & \infty & 0 & 4 \\
F & \infty & \infty & 8 & \infty & 4 & 0
\end{array}
$$

比较大小



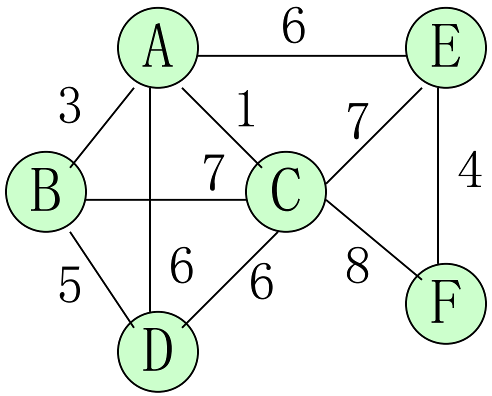
V-U中各顶点到U的最短直接路径LOWCOST[i]：

0	3	1	5	6	8
0	3	1	5	6	4

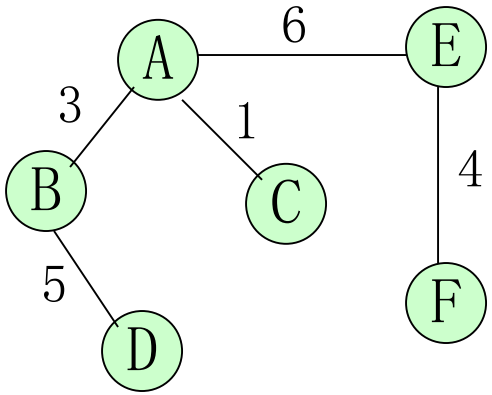
A	B	C	D	E	F
A	A	A	B	A	C
A	A	A	B	A	E

相邻顶点CLOSEST[i]：

Prime算法：



图G



V-U中各顶点到U的最短直接路径LOWCOST[i]：

0	3	1	5	6	4

↓

A	B	C	D	E	F
0	3	1	5	6	4

A	A	A	B	A	E

↓

相邻顶点CLOSEST[i]：

A	A	A	B	A	E

7.4 最小生成树算法

- 普里姆（Prim）算法的实现
  - 数据结构
    - 数组LOWCOST[n]：用来保存集合V-U中各顶点与集合U中顶点最短边的权值，LOWCOST[v]=0表示顶点v已加入最小生成树中；
    - 数组CLOSEST[n]：用来保存依附于该边的（集合V-U中各顶点与集合U中顶点的最短边）在集合U中的顶点。
  - 如何用数组LOWCOST[n]和CLOSEST[n]表示候选最短边集？
    - LOWCOST[i]=w
    - CLOSEST[i]=k

    } **表示顶点v_i和顶点v_k之间的权值为w，其中：$v_i \in$ V-U 且$v_k \in$ U**
  - 如何更新？

$$\begin{cases} \textbf{LOWCOST[i]=min \{ cost}\,(v_k,\ v_i)\ |\ v_k \in \textbf{U, LOWCOST[i]\}} \\ \textbf{CLOSEST[i]=}k \end{cases}$$

7.4 最小生成树算法

- 实现步骤:

1. 初始化两个辅助数组LOWCOST和CLOSEST;
2. 输出顶点v_0，将顶点v_0加入集合U中;
3. 重复执行下列操作n-1次

   3.1 在LOWCOST中选取最短边，取CLOSEST中对应的顶点序号k;

   3.2 输出顶点k和对应的权值;

   3.3 将顶点k加入集合U中;

   3.4 调整数组LOWCOST和CLOSEST;

7.4 最小生成树算法

- 普里姆（Prim）算法的实现

```
void  Prim(Costtype  C[n+1][n+1]  )
{ costtype  LOWCOST[n+1];   int  CLOSEST[n+1];   int i,j,k;  costtype  min;
     for( i=2; i<=n; i++ )
     {     LOWCOST[i] = C[1][i];      CLOSEST[i] = 1;    }
     for(  i = 2;  i <= n; i++ )
     {      min = LOWCOST[i];
            k = i;
            for(  j = 2;   j <= n;  j++ )
                  if ( LOWCOST[j] < min )
                   {  min = LOWCOST[j] ;     k=j;  }
            cout << "("  << k << "," << CLOSEST[k] <<  ")"  <<  end1;
            LOWCOST[k] = 0 ;
            for ( j = 2;  j <= n;  j++ )
                 if  ( C[k][j] <  LOWCOST[j])
                 {      LOWCOST[j]=C[k][j];   CLOSEST[j]=k;     }
     }
} /* 时间复杂度:  O( |V|^2 )
```

7.4.2 最小生成树（Minimun spanning tree ）

- **Kruskal 算法**

克鲁斯卡尔算法的基本思想：

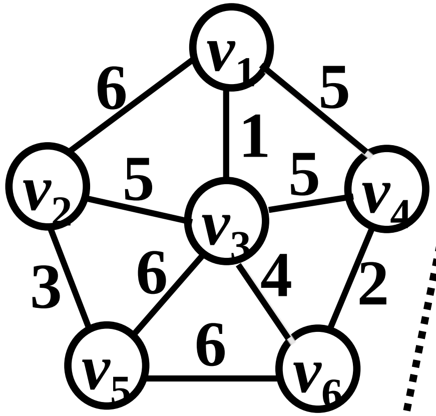
考虑问题的出发点：为使生成树上边的权值之和达到最小，则应使生成树中每一条边的权值尽可能地小。

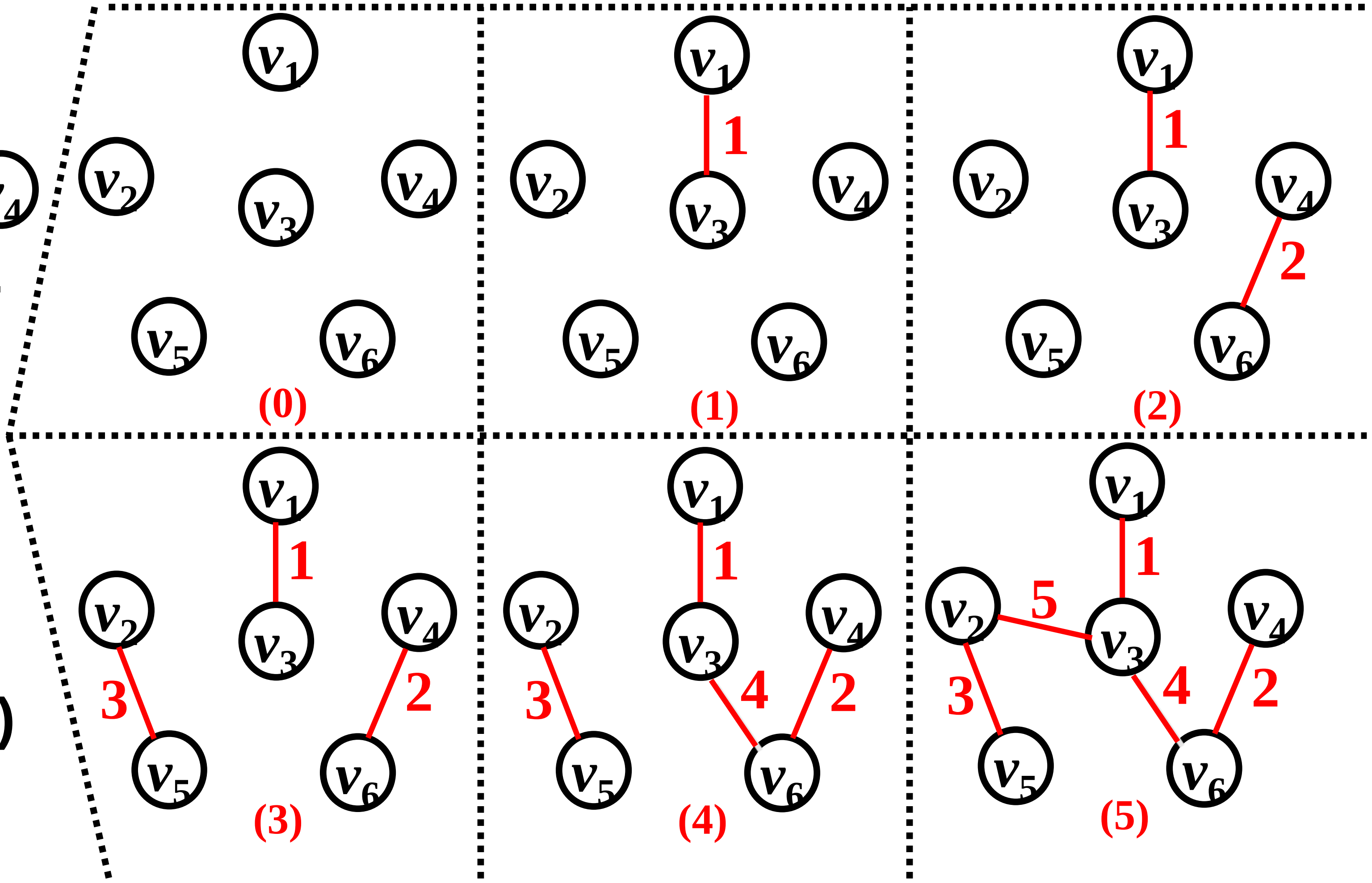
7.4 最小生成树算法

- 克鲁斯卡尔（Kruskal）算法
  - 基本思想：
    - 设无向连通网为G＝(V, E)，令G的最小生成树为T＝(U, TE)，其初态为U＝V，TE＝{ }，
    - 然后，按照边的权值由小到大的顺序，依次考察G的边集E中的各条边。
    - 若被考察的边连接的是两个不同连通分量，则将此边作为最小生成树的边加入到T中，同时把两个连通分量连接为一个连通分量；
    - 若被考察的边连接的是同一个连通分量，则舍去此边，以免造成回路，
    - 如此下去，当T中的连通分量个数为1时，此连通分量便为G的一棵最小生成树。

7.4 最小生成树算法

(v_1,v_3)	(v_4,v_6)	(v_2,v_5)	$(v3,v_6)$	(v_1,v_4)	(v_3,v_4)	(v_2,v_3)	(v_1,v_2)	(v_3,v_5)	(v_5,v_6)
1	2	3	4	5	5	5	6	6	6





O (|E|*log|E|)

7.4 最小生成树算法

- 克鲁斯卡尔（Kruskal）算法
  - 实现步骤：

1. 初始化：U=V； TE={ }；

2. 循环直到T中的连通分量个数为1

   2.1 在E中选择最短边(u，v);

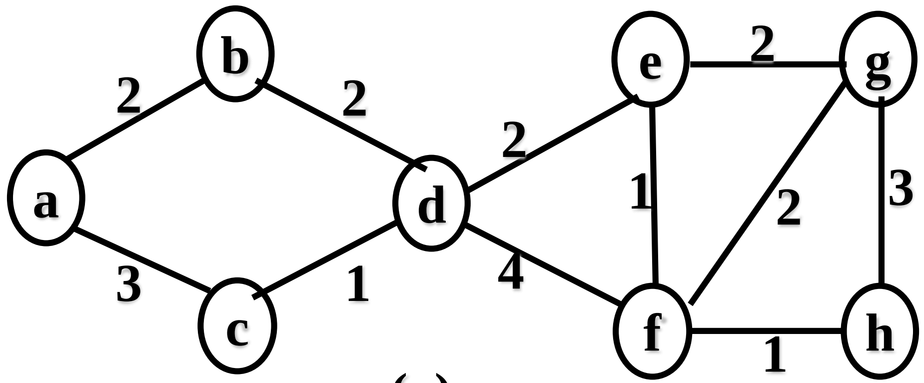
   2.2 如果顶点u、v位于T的两个不同连通分量，则

      2.2.1 将边(u，v)并入TE；

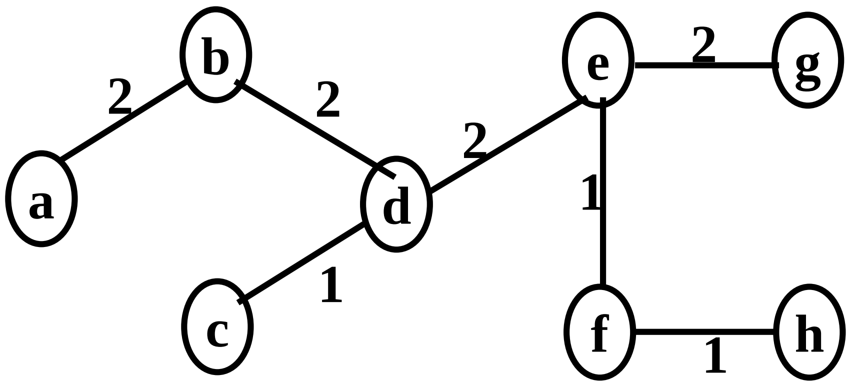
      2.2.2 将这两个连通分量合为一个；

   2.3 在E中标记边(u，v)，使得(u，v)不参加后续最短边的选取

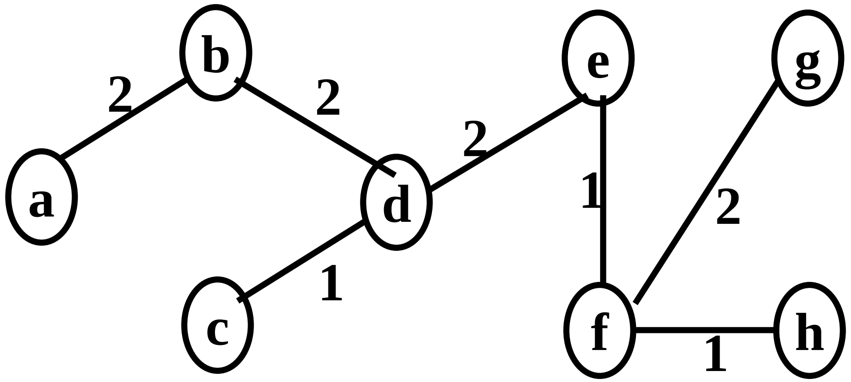
例



(a)



(b)



(c)

当各边有相同权值时，由于选择的任意性，产生的生成树可能不唯一

当各边的权值不相同时，产生的生成树是唯一的。

算法比较

***Prim* 算法（普里姆）**： 从某一个顶点开始构建生成树； 每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

时间复杂度：$O(|V|^2)$ 适合用于边稠密图

***Kruskal* 算法（克鲁斯卡尔）**： 每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选） 直到所有结点都连通

时间复杂度：$O(|E|*\log|E|)$ 适合用于边稀疏图