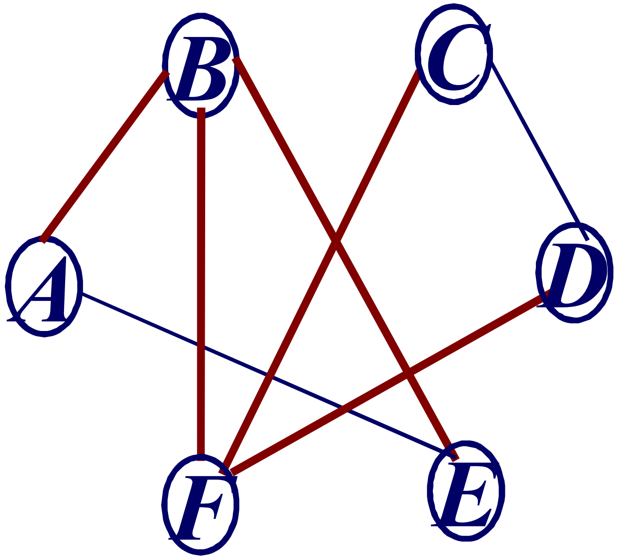
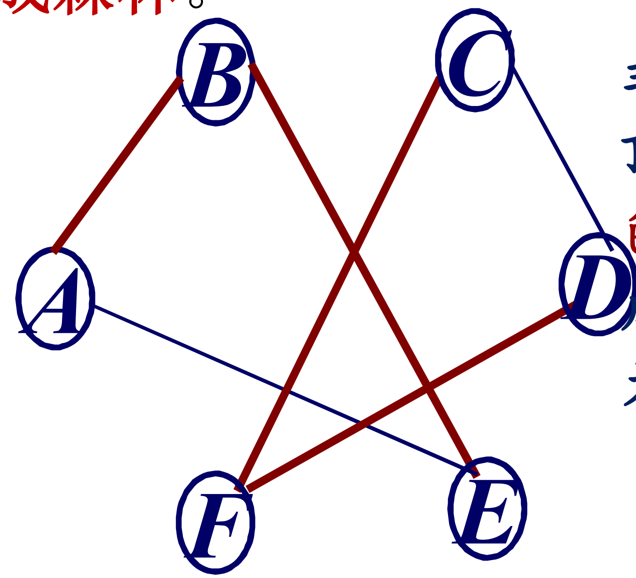
生成树

- 生成树：假设一个连通图有 n 个顶点和 e 条边，其中 n-1 条边和 n 个顶点构成一个极小连通子图，称该极小连通子图为此连通图的生成树。

- 生成森林：对非连通图，则称由各个连通分量的生成树的集合为此非连通图的生成森林。

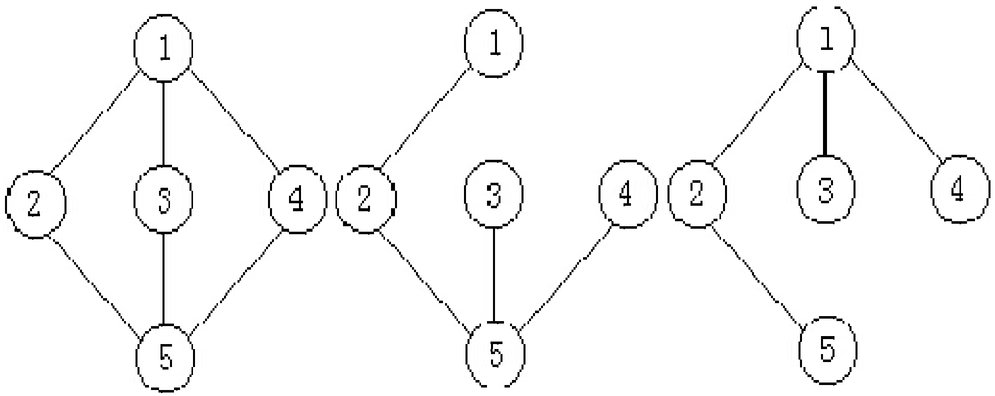
连通图，顶点和紫色连边构成的子图为其生成树





非连通图，顶点和紫色连边构成的子图为其生成森林

生成树是否唯一?



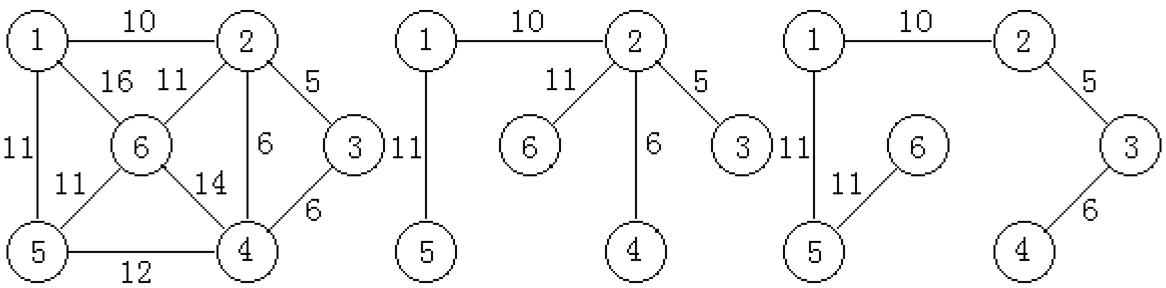
G　　T1　　T2

根据定义不保证唯一性

图G至少有2棵生成树T1和T2

最小生成树

- 最小生成树：带权图的生成树上的各边权值之和称为这棵树的代价。最小代价生成树是各边权值的总和最小的生成树。



（a）一个网络　（b）最小生成树　（c）最小生成树

最小生成树----问题的应用背景

- 例如：以尽可能低的总造价建造城市间的通讯网络，把十个城市联系在一起。在这十个城市中，任意两个城市之间都可以建造通讯线路，通讯线路的造价依据城市间的距离不同而有不同的造价，可以构造一个通讯线路造价网络，在网络中，每个顶点表示城市，顶点之间的边表示城市之间可构造通讯线路，每条边的权值表示该条通讯线路的造价，要想使总的造价最低，实际上就是寻找该网络的**最小生成树**。

最小生成树构造方法

- prim普里姆
- Kruskal克鲁斯卡尔

最小生成树

- **MST性质(最小生成树性质):**

  令G=(V,E,W)为一个带权连通图，T为G的一生成树。对任一不在T中的边*uv*，如果将*uv*加入T中会产生一回路，使得*uv*是回路中权值最大的边。那么树T具有MST性质。

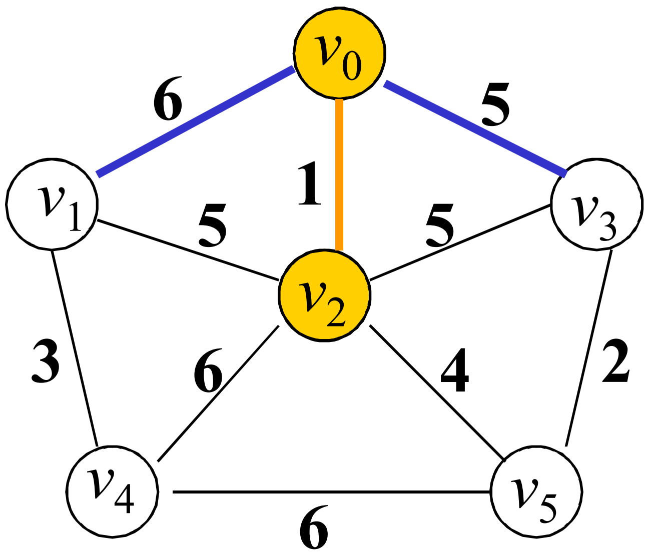
Prim算法的基本步骤

- **G=(V, E)是连通网，TE是G上最小生成树中边的集合。**
- **初态：U={u_0|u_0∈V}, TE={}开始，重复执行下述操作：**

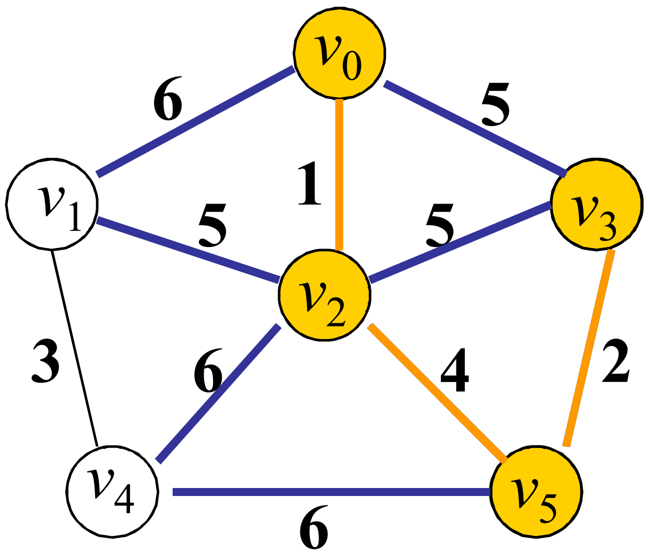
1）在所有u∈U，v∈V-U的边(u,v)∈E中找一条带权最小的边(u_0,v_0)并入集合TE, v_0并入U。

2）重复1）直至U=V为止。此时TE中必有n-1条边，则T=（V，TE）为G的最小生成树。

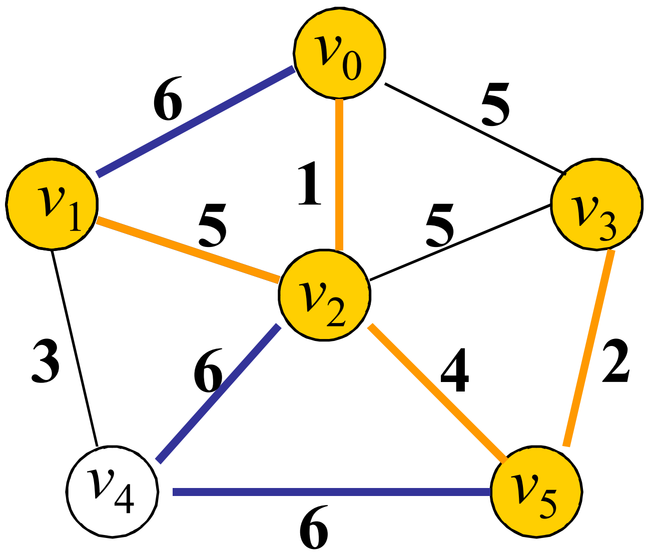
Prim算法



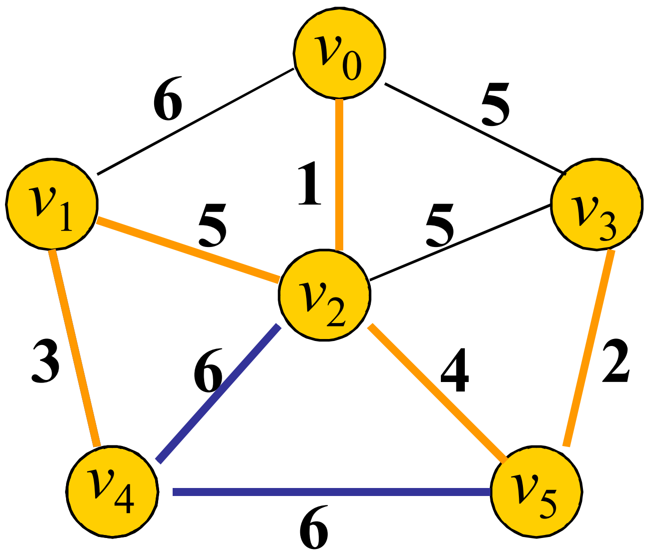
Prim算法



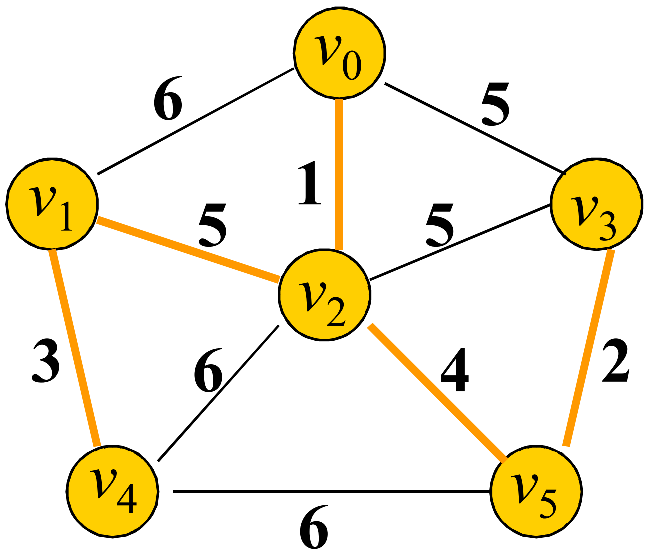
Prim算法

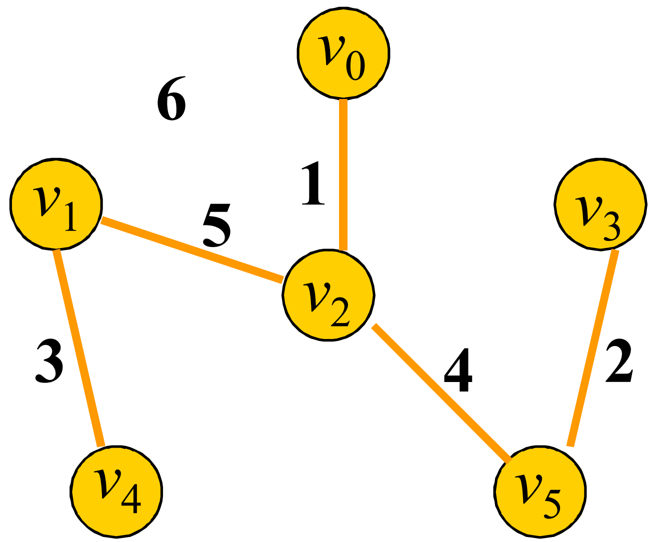


Prim算法

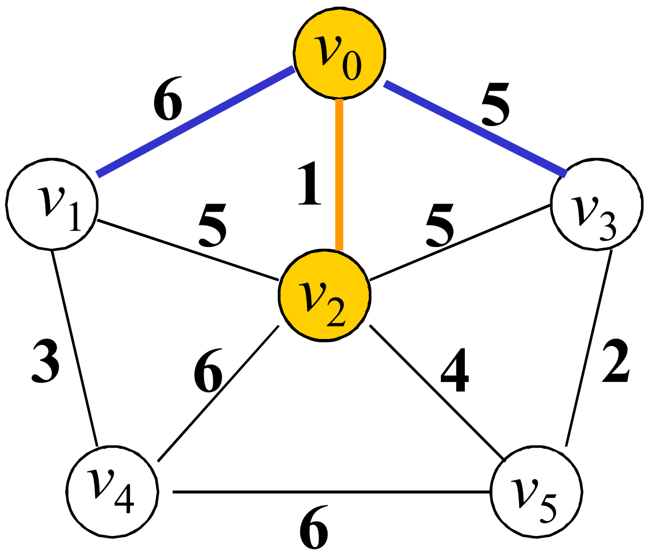


Prim算法

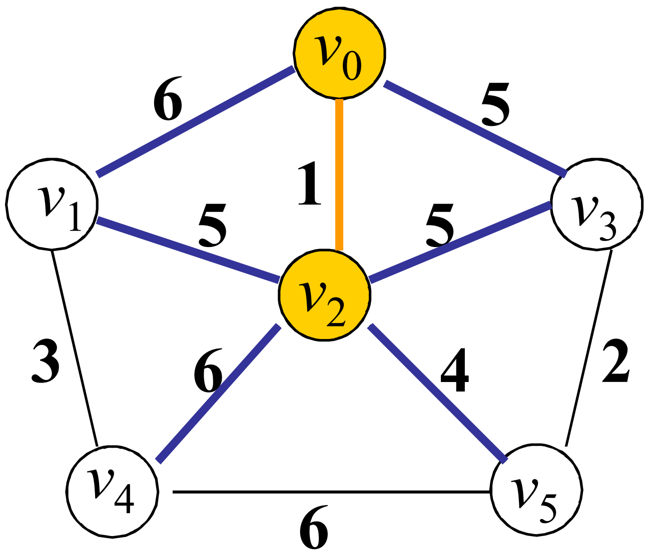




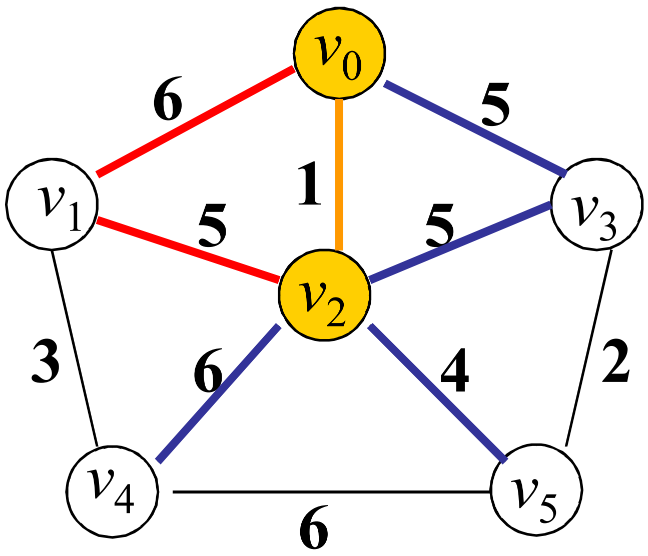
Prim算法



Prim算法

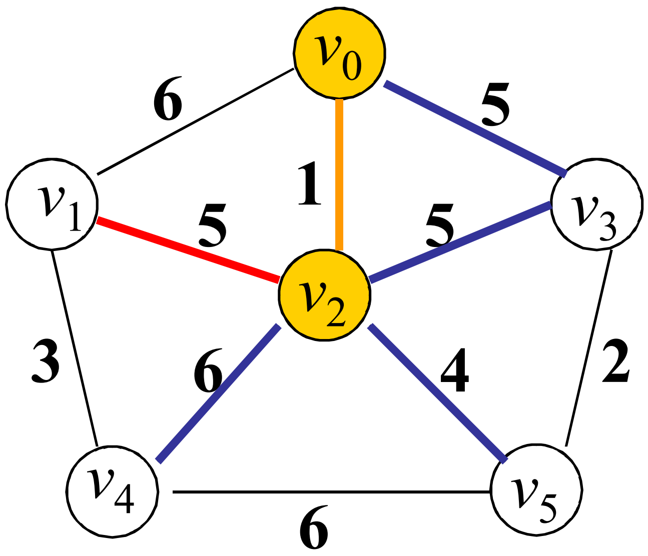


Prim算法



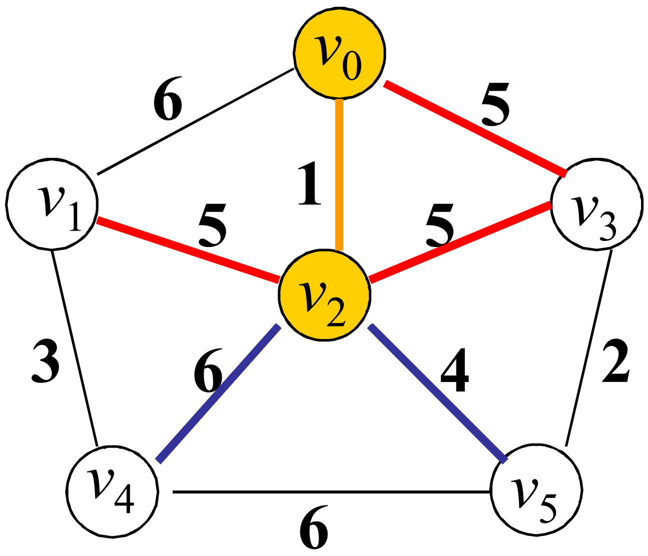
保存每个不在生成树中的点与生成树相连的**最好**情况：和生成树中那个顶点相连，连边的权值

Prim算法



保存每个不在生成树中的点与生成树相连的**最好**情况：和生成树中那个顶点相连，连边的权值

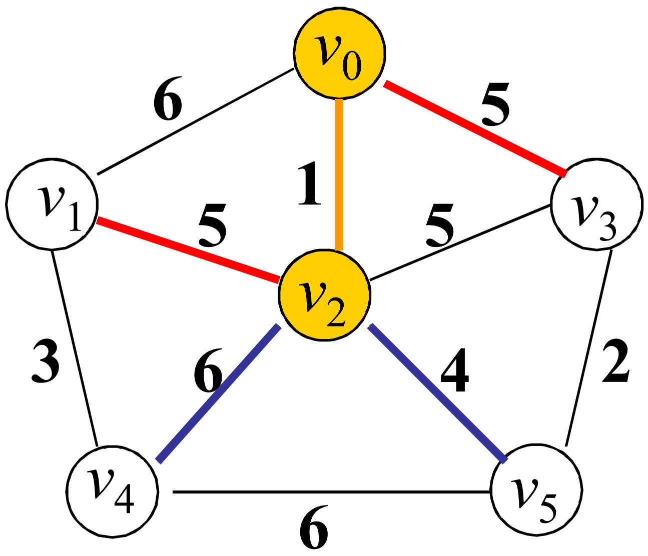
Prim算法



保存每个不在生成树中的点与生成树相连的**最好**情况：和生成树中那个顶点相连，连边的权值

Prim算法

保存每个不在生成树中的点与生成树相连的**最好**情况：和生成树中那个顶点相连，连边的权值

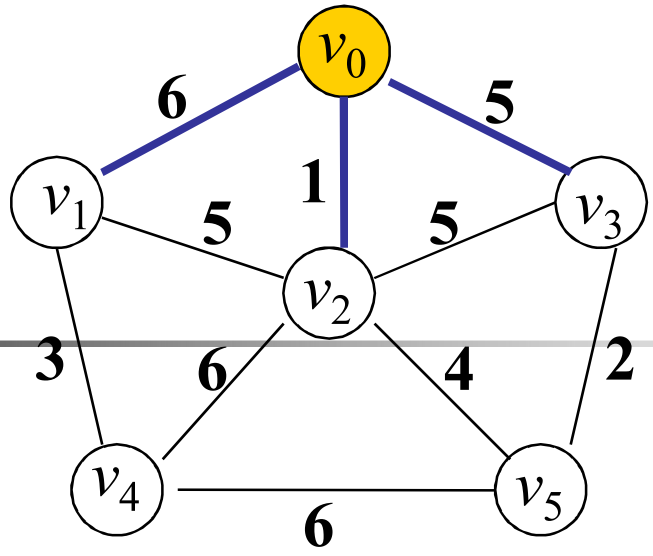


Prim算法

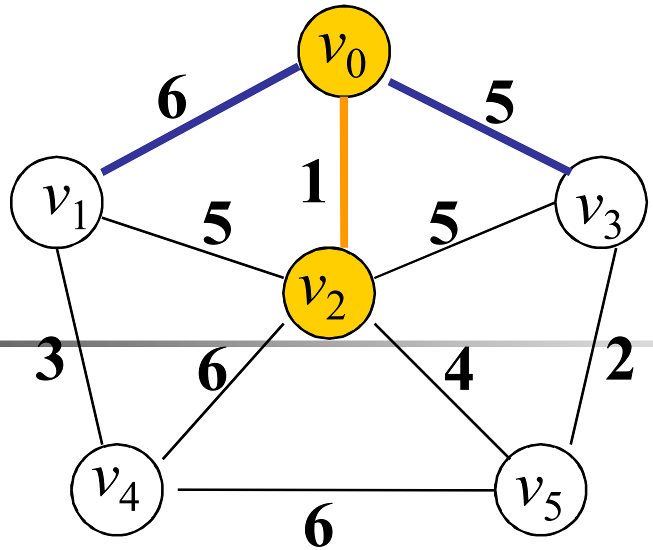
- 采用一维数组closedge[MAX]存放图中每个顶点与生成树中顶点相连的**最好**情况：
- typedef struct{

```
typedef struct{
        int adjvex;
        int lowcost;}EdgeType;
```

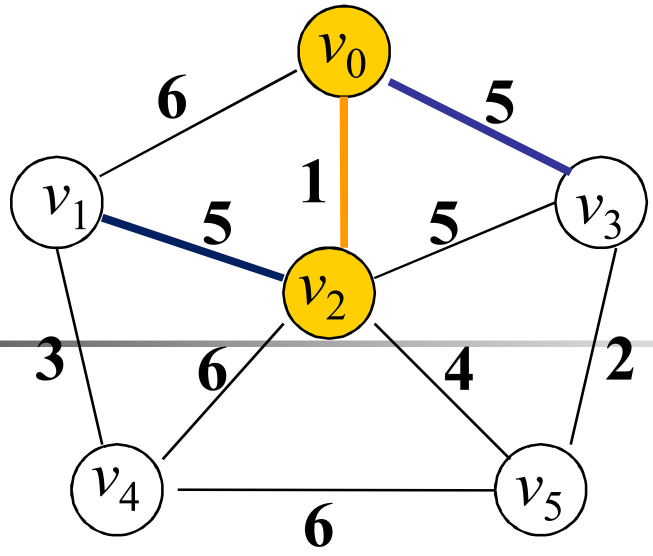
- EdgeType closedge[MAX];
  - 当顶点v尚未加入生成树时，closedge[v]存放的是v与生成树中的顶点相连的**最好**情况：v与生成树中的顶点的所有连边中权值最小的那条边；　closedge[v].adjvex存放的这条权值最小的边的另一个顶点，closedge[v].lowcost存放的这条权值最小的边的权值
- 如何区分生成树中的顶点和不在生成树中的顶点呢?
  - closedge[w].lowcost==0表示w已经加入生成树



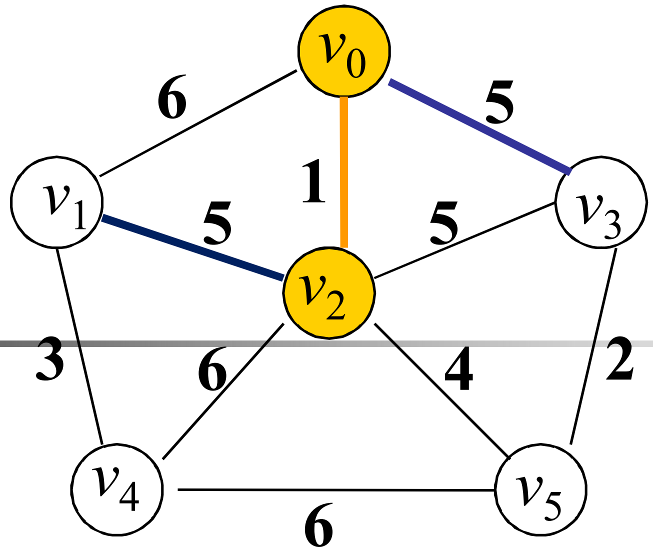
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2



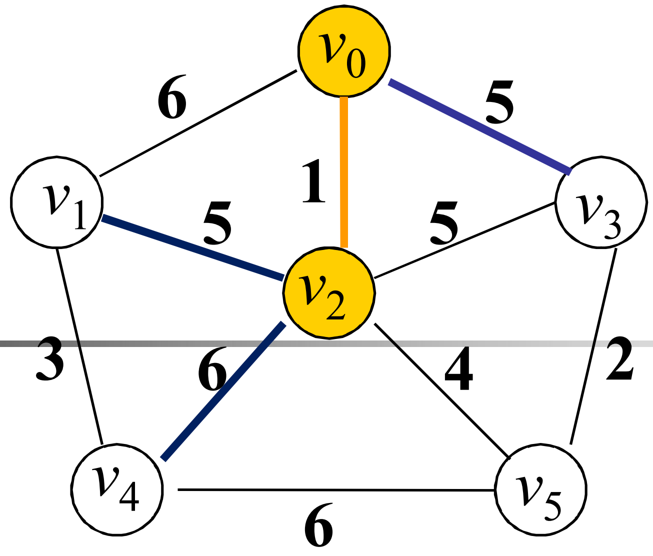
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***		v_0 1				$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



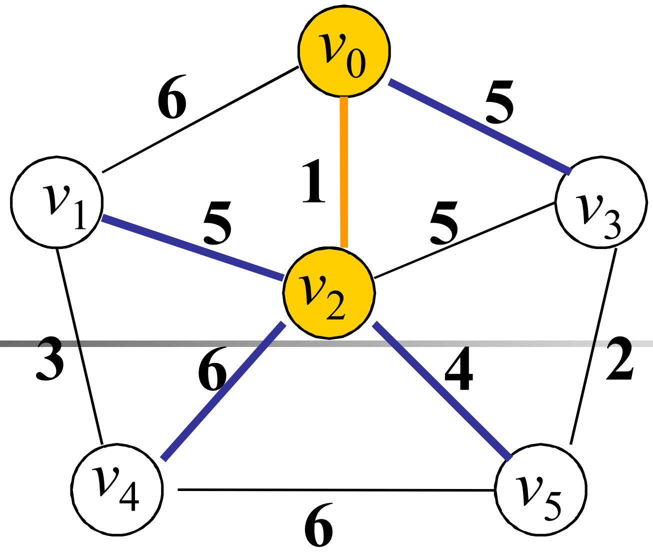
	1	2	3	4	5	U	V-U	k
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 1				$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



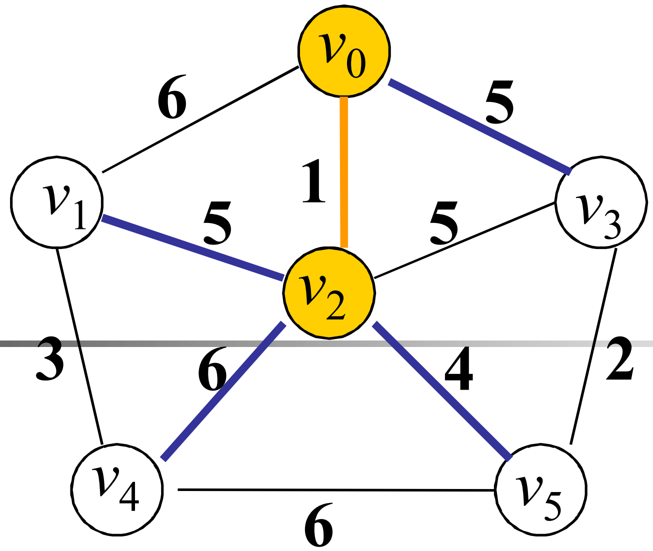
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 1	v_0 5			$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



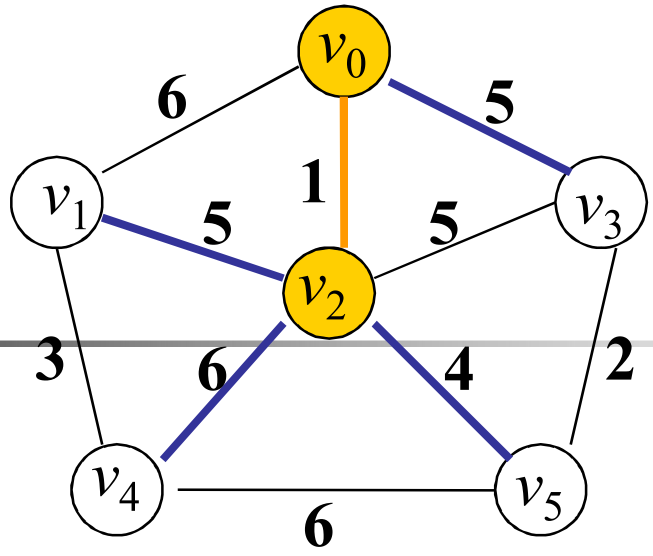
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 1	v_0 5	v_2 6		$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



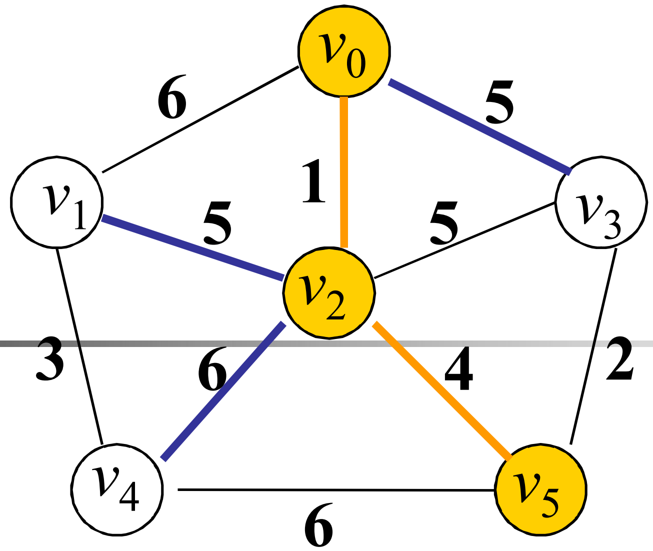
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 1	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



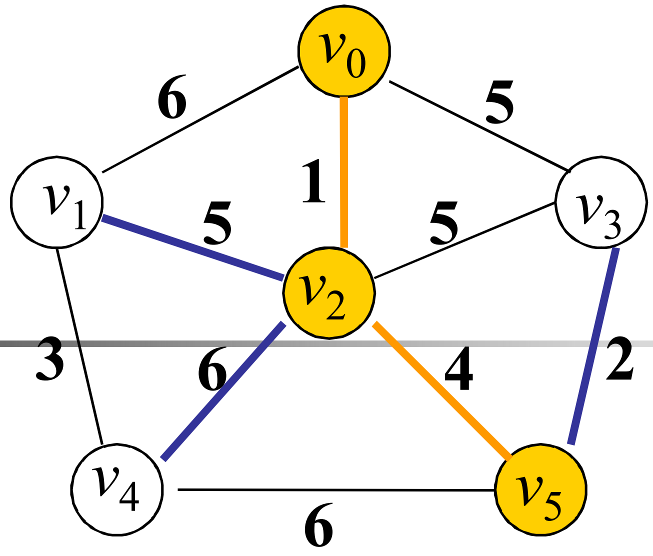
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



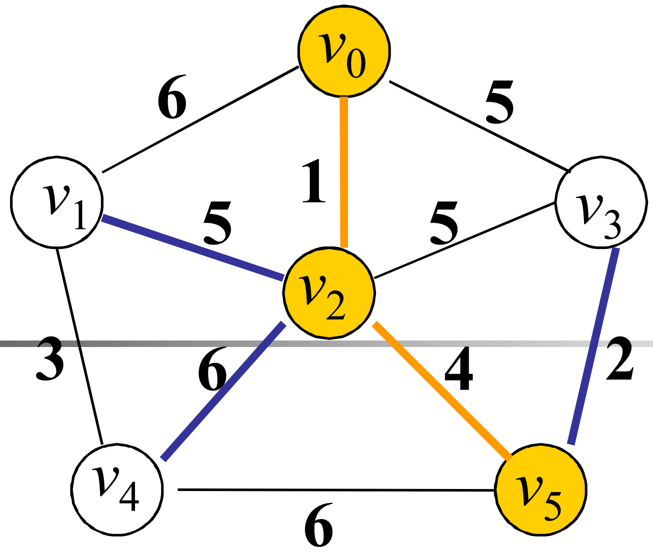
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5



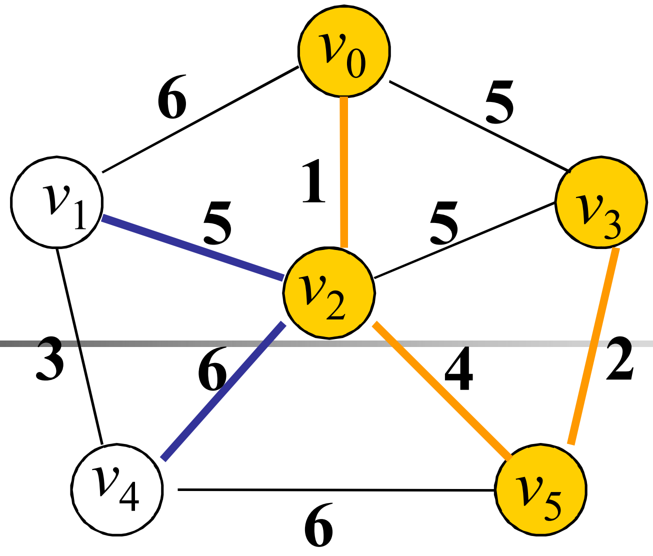
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	



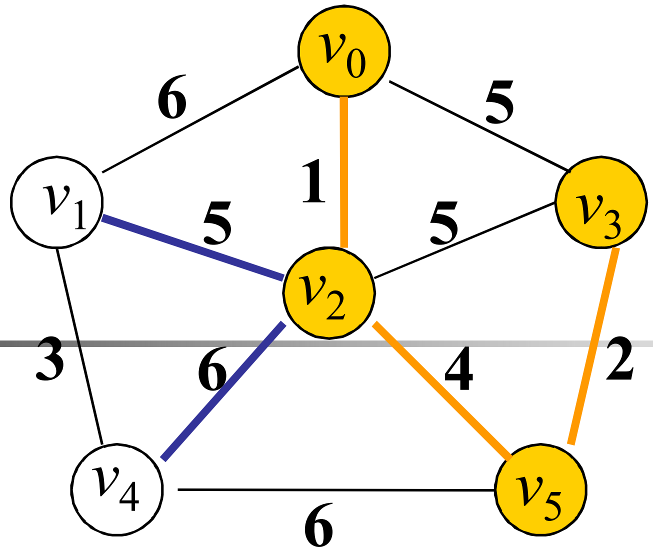
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	



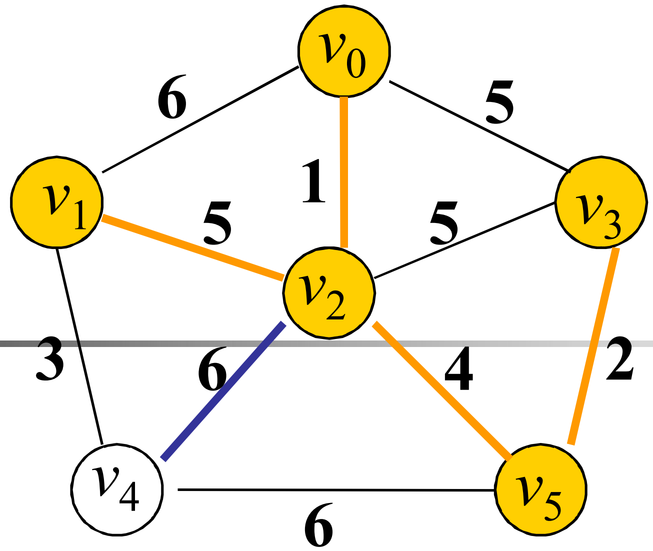
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3



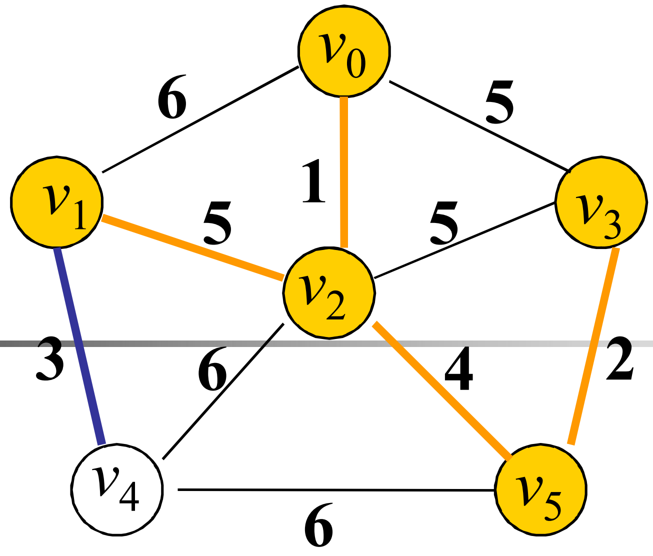
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex ***lowcost***	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex ***lowcost***	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	



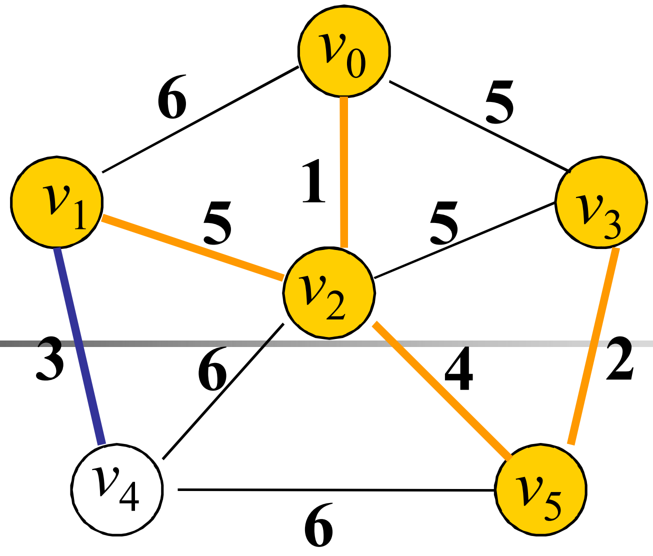
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	1



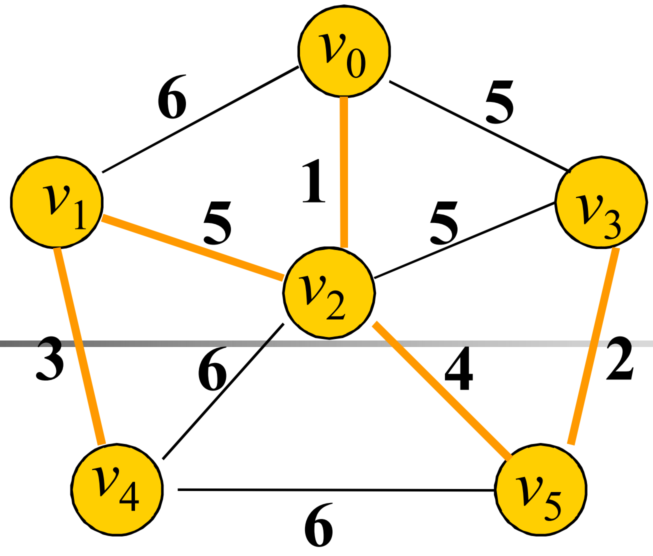
	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex ***lowcost***	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex ***lowcost***	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	1
adjvex ***lowcost***	v_2 0	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3, v_1\}$	$\{v_4\}$	



	1	2	3	4	5	U	V-U	*k*
adjvex ***lowcost***	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex ***lowcost***	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex ***lowcost***	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex ***lowcost***	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	1
adjvex ***lowcost***	v_2 0	v_0 0	v_5 0	v_1 3	v_2 0	$\{v_0, v_2, v_5, v_3, v_1\}$	$\{v_4\}$	



	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	1
adjvex *lowcost*	v_2 0	v_0 0	v_5 0	v_1 3	v_2 0	$\{v_0, v_2, v_5, v_3, v_1\}$	$\{v_4\}$	4



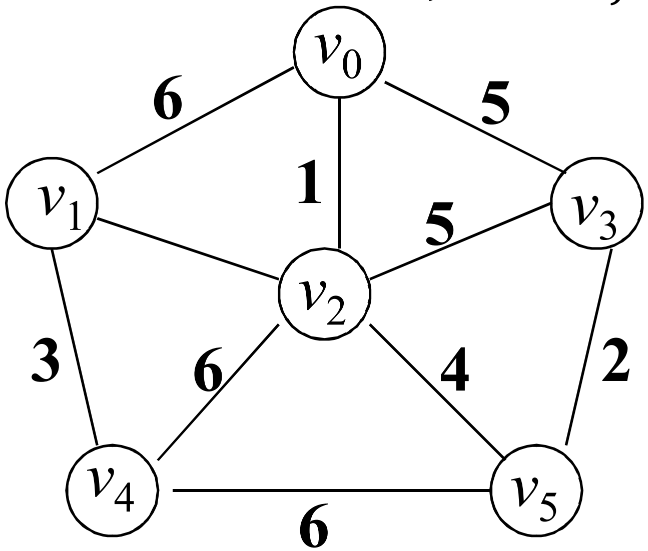
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*	v_2 5	v_0 0	v_5 0	v_2 6	v_2 0	$\{v_0, v_2, v_5, v_3\}$	$\{v_1, v_4\}$	1
adjvex *lowcost*	v_2 0	v_0 0	v_5 0	v_1 3	v_2 0	$\{v_0, v_2, v_5, v_3, v_1\}$	$\{v_4\}$	4
adjvex *lowcost*	v_2 0	v_0 0	v_5 0	v_1 0	v_2 0	$\{v_0, v_2, v_5, v_3, v_1, v_4\}$		

Prim算法

- 图采用邻接矩阵和邻接表存放，下面以邻接矩阵为例实现prim算法
- define MAX 100
- define MAXEDGE 1000000
- typedef struct{

  int arcs[MAX][MAX];

  int vexnum,arcnum;}AGraphs;



$$
\begin{bmatrix}
0 & 6 & 1 & 5 & \infty & \infty \\
6 & 0 & 5 & \infty & 3 & \infty \\
1 & 5 & 0 & 5 & 6 & 4 \\
5 & \infty & 5 & 0 & \infty & 2 \\
\infty & 3 & 6 & \infty & 0 & 6 \\
\infty & \infty & 4 & 2 & 6 & 0
\end{bmatrix}
$$

```
void prim(AGraphs G,int u)
 {  int i,j,k;
    EdgeType closedge[MAX];
     for(j=0;j<G.vexnum;j++)
         {  closedge[j]. adjvex=u;   closedge[j]. lowcost=G.arcs[u][j];}
     closedge[u]. lowcost=0;
     for(i=1;i<G.vexnum;i++)
    {  k=minclosedge(closedge);
       printf("(%d,%d)", closedge[k]. adjvex,k);
       closedge[k]. lowcost=0;
       for(j=0;j<G.venum;j++)
           if(G.arcs[k][j]< closedge[j]. lowcost)
           {   closedge[j]. lowcost= G.arcs[k][j];
               closedge[j]. adjvex =k;
            }
    }
 }
```

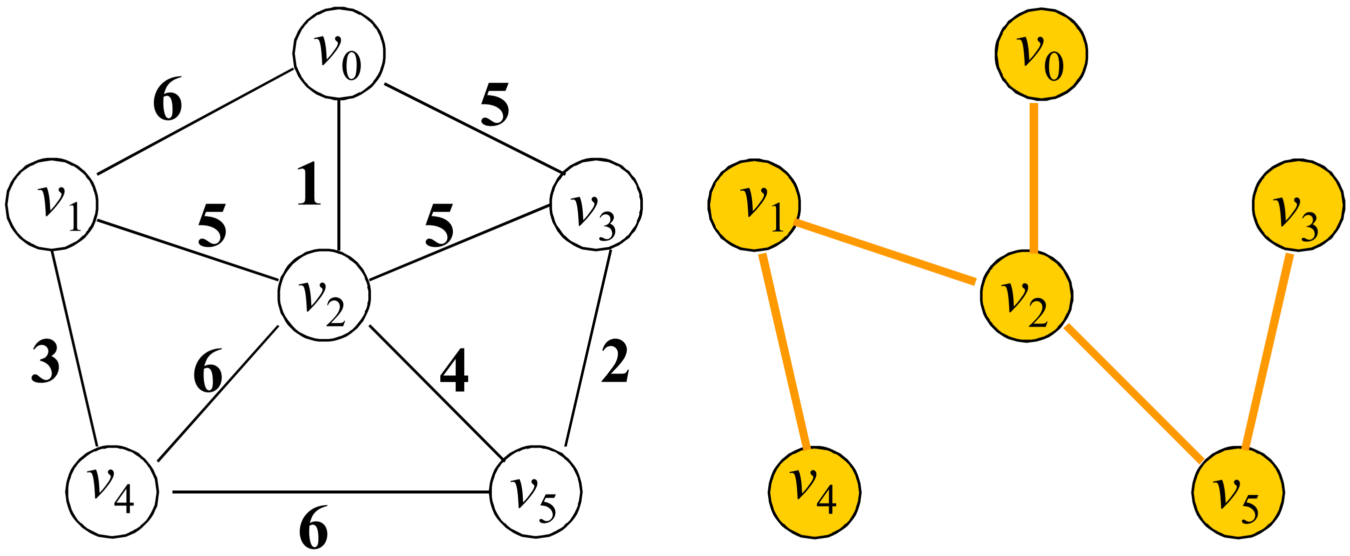
```
int minclosedge(EdgeType closedge[ ])
 {  int min,j,k;
   min=MAXEDGE;
   k=-1;
   for(j=0;j<G.vexnum;j++)
        if (closedge[j]. lowcost !=0&&closedge[j]. lowcost<min)
         {
              min=closedge[j]. lowcost;
              k=j;
          }
    return k;
  }
```

时间复杂度：$O(n^2)$

Prim算法适合于稠密图

Kruskal算法的基本步骤

- 设G＝(V,E)，T为G的最小生成树，初态T＝(V,{})
- 按照边的权值由小到大的顺序，考察G的边集E中的各条边。若被考察的边的两个顶点属于T的两个不同的连通分量，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；若被考察边的两个顶点属于同一个连通分量，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。



Kruscal算法适合稀疏图，时间复杂度为O(eloge)，
e为图的边数