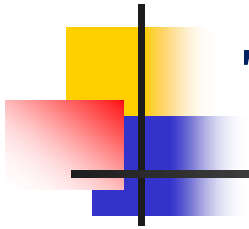
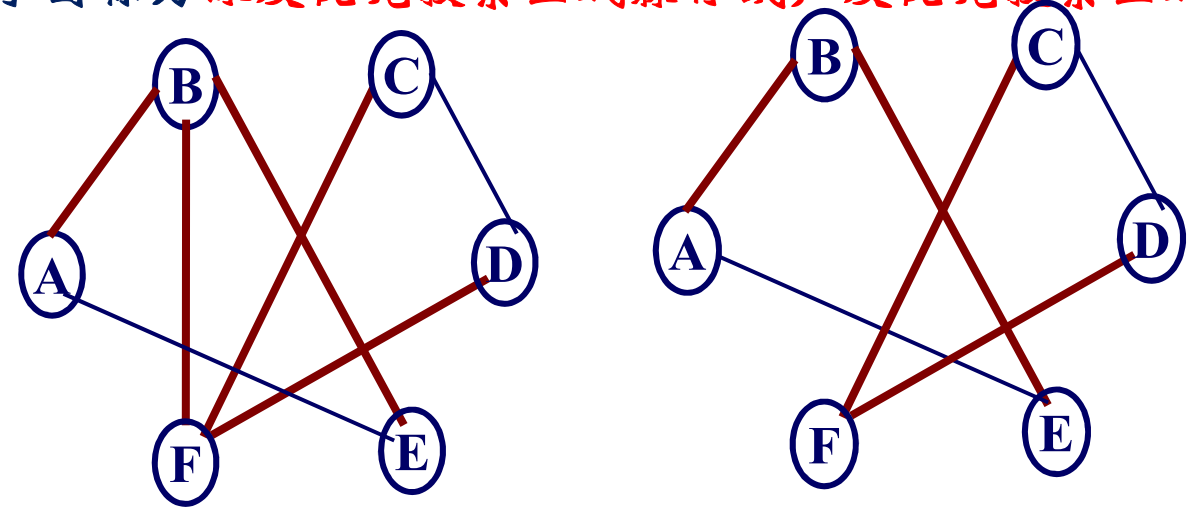


7.4 图的连通性问题

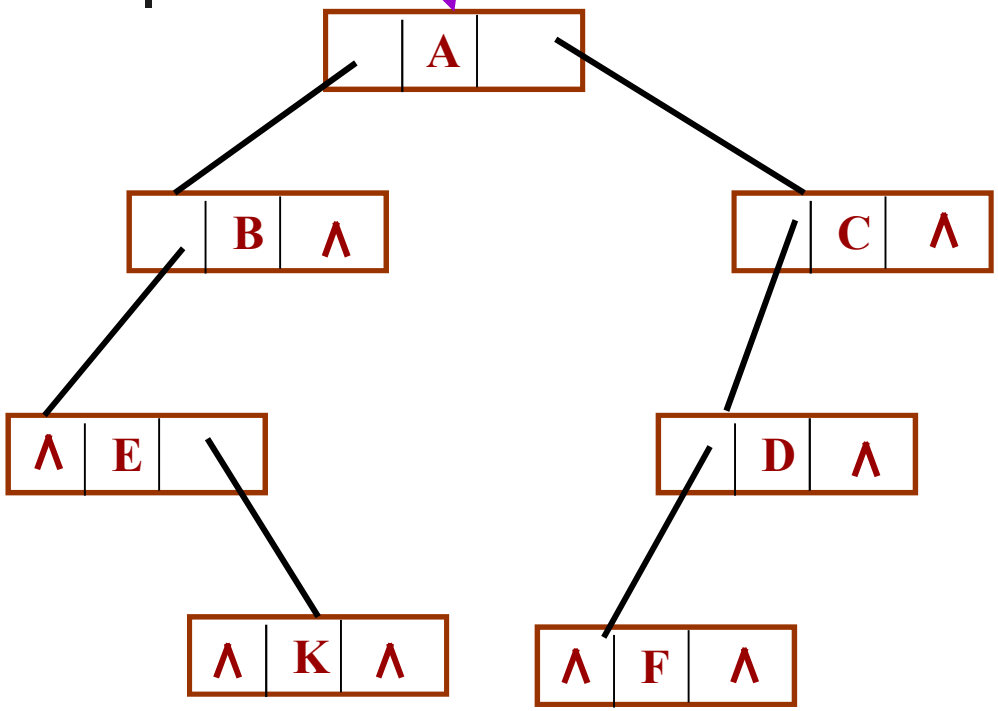
- 无向图的连通分量和生成树
- 最小生成树:
- 说明：本节研究的是无向图

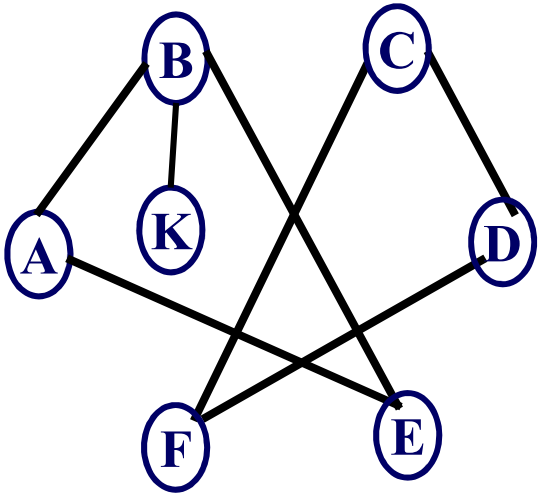
7.4 图的连通性问题--连通分量和生成树

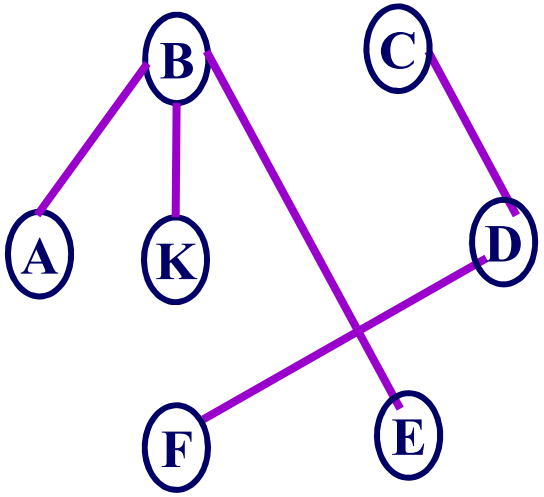
- 若从无向图中任一点出发采用深度优先搜索或广度优先搜索能访问到图中所有顶点，则该图为连通图，否则为非连通图
- 对连通图：深度优先搜索或广度优先搜索访问时经过的顶点和边构成的子图称为深度优先搜索生成树或广度优先搜索生成树
- 对非连通图：深度优先搜索或广度优先搜索访问时经过的顶点和边构成的子图称为深度优先搜索生成森林或广度优先搜索生成森林



7.4 图的连通性问题--连通分量和生成树

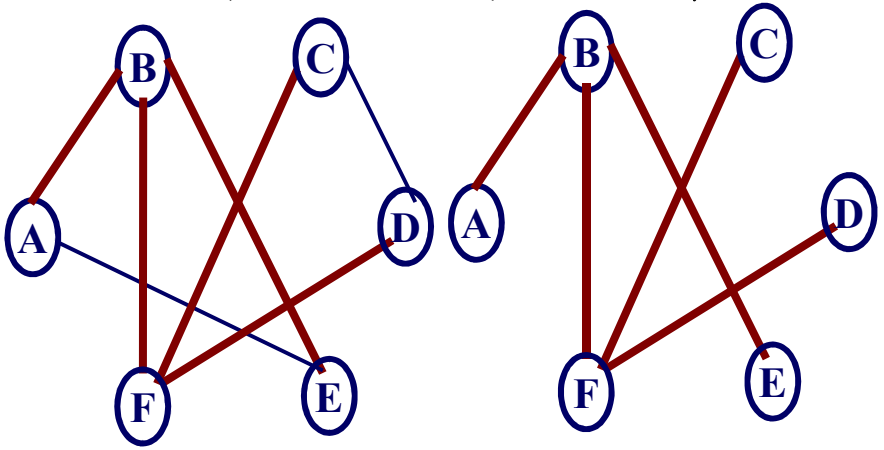




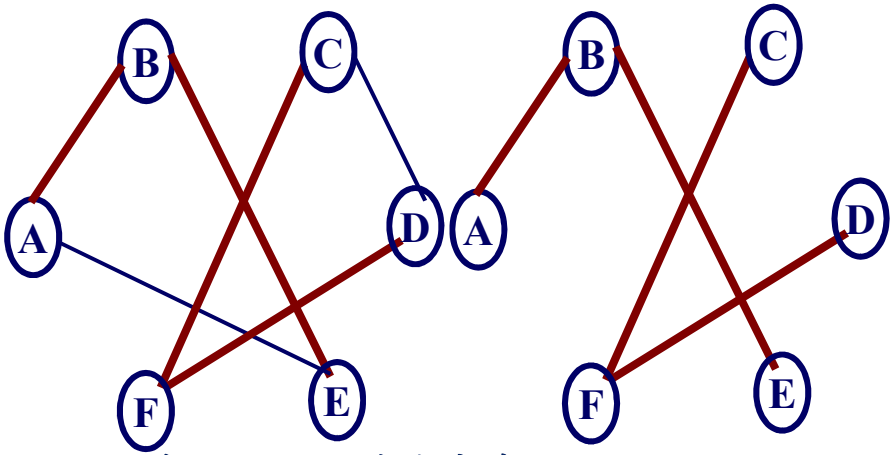


生成树

- 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。

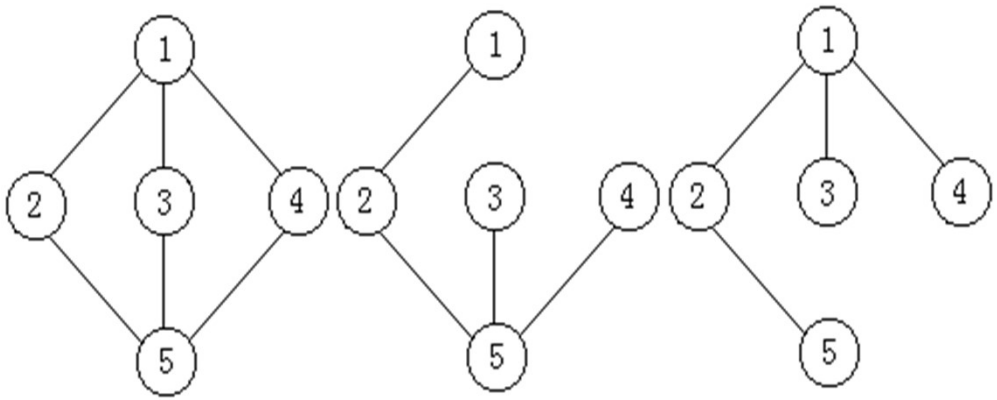


连通图及其生成树示例
顶点和紫色连边构成的子图为该图的生成树



非连通图及其生成森林示例
顶点和紫色连边构成的子图为该图的生成森林

生成树是否唯一?



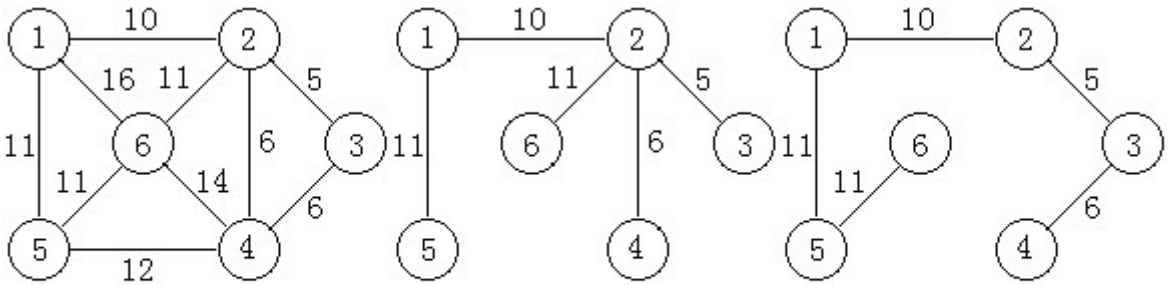
G　　T1　　T2

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。



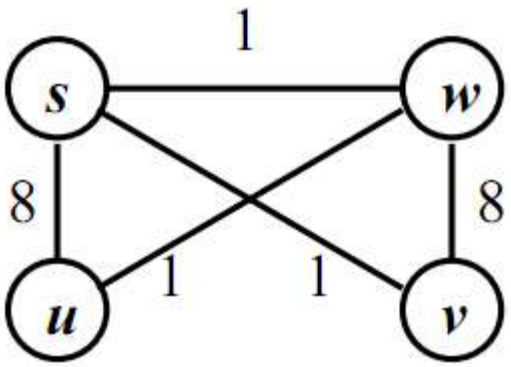
根据定义不保证一个带权图的最小生成树是唯一的

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

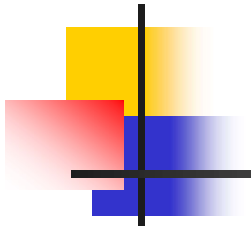
- **问题：图的深度优先搜索是否能得到最小生成树?**



对该图从顶点*s*出发分别采用深度优先搜索和广度优先搜索小生成树是否能得到该图的最小生成树?

- 构造最小生成树的算法:

1. prim普里姆
2. Kruskal克鲁斯卡尔



最小生成树

- MST性质(最小生成树性质):

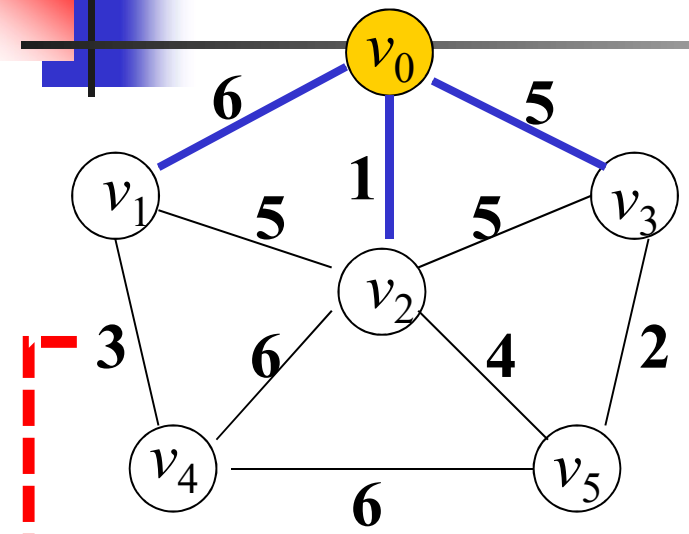
  令G=(V, E, W)为一个带权连通图，T为G的一生成树。对任一不在T中的边uv，如果将uv加入T中会产生一回路，使得uv是回路中权值最大的边。那么树T具有MST性质。

Prim算法的基本步骤

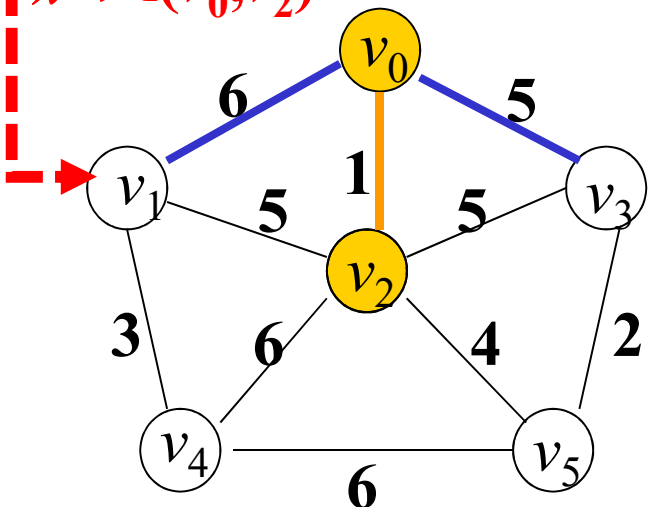
- G=(V, E)是连通网，生成树T=(U,TE)//TE是G上最小生成树中边的集合。
- 操作步骤：

1. 令U={u_0|u_0∈V}, TE={}；
2. 在所有u∈U，v∈V-U的边(u,v)∈E中找一条权值最小的边(u_0,v_0)并入集合TE, v_0并入U;
3. 重复步骤2，直至U=V为止;

//此时TE中有n-1条边，T=（V, TE）为G的最小生成树



加入(v_0,v_2)



初始：任选一个顶点出发构造最小生成树，假设选v_0为出发点，将v_0加入当前的最小生成树T，当前T中只有一个顶点，0条边，即T=(U,TE)，U={v_0}，TE={}。（见左侧上图）

之后重复做n-1步，每步选一个顶点一条边加入生成树。

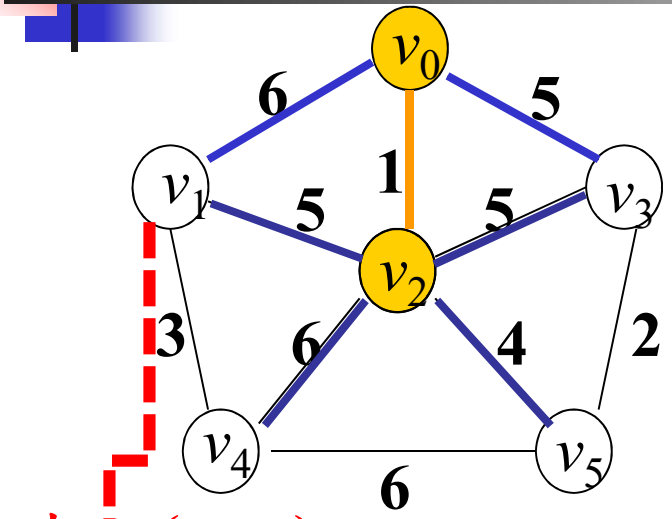
第一步：从蓝色边中选一条权值最小的边(v_0,v_2)加入生成树（见左侧下图）

说明：蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边

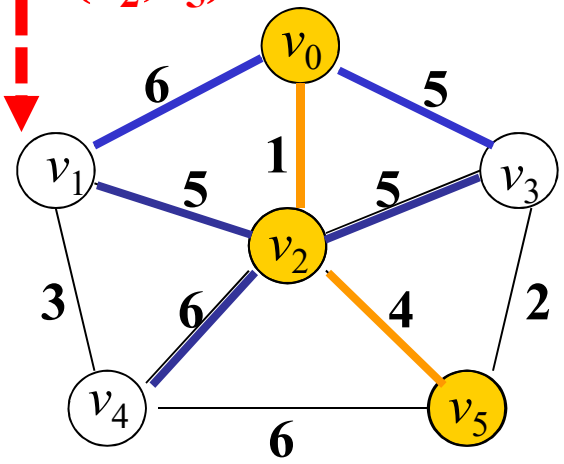
Prim算法求解过程演示

第二步：从（左侧上图）蓝色边中选一条权值最小的边(v_2,v_5)加入生成树（左侧下图）

说明：蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边



加入(v_2,v_5)

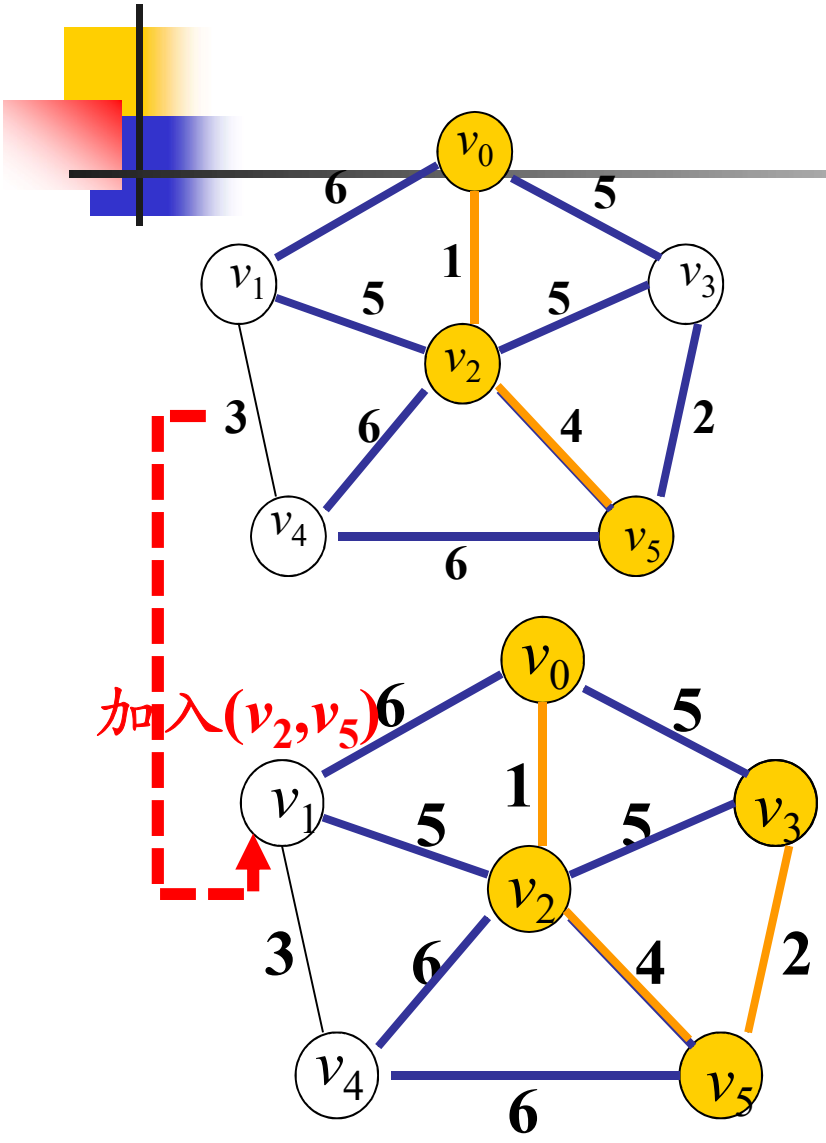


加入2条边后，生成树

T=(U,TE)，U={v_0, v_2,v_5}，

TE={(v_0,v_2)，(v_2,v_5)}

Prim算法求解过程演示



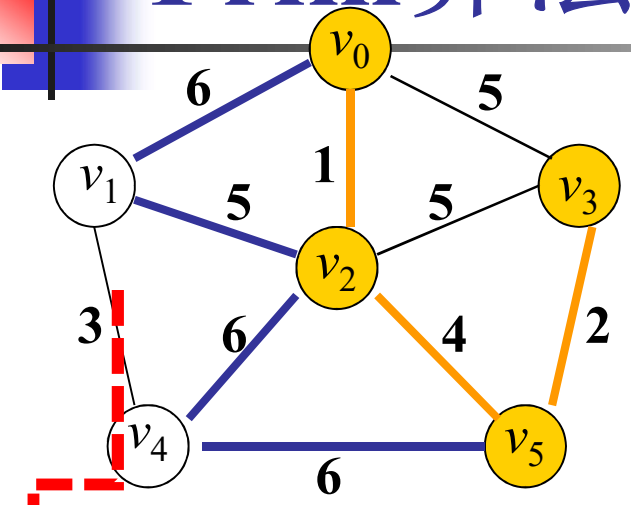
第三步：从（左侧上图）蓝色边中选一条权值最小的边(v_5,v_3)加入生成树（左侧下图）

说明：蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边

加入3条边后，生成树
T=(U,TE)，U={v_0, v_2,v_5 ,v_3},
TE={(v_0,v_2), (v_2,v_5), (v_5,v_3)}

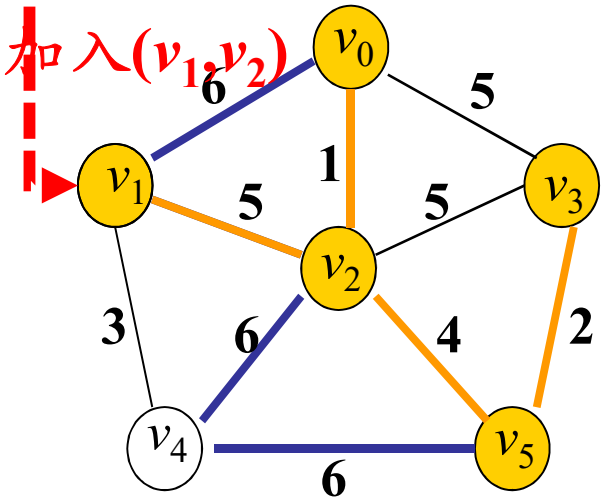
Prim算法求解过程演示

Prim算法



第四步：从（左侧上图）蓝色边中选一条权值最小的边(v_1,v_2)加入生成树（左侧下图）

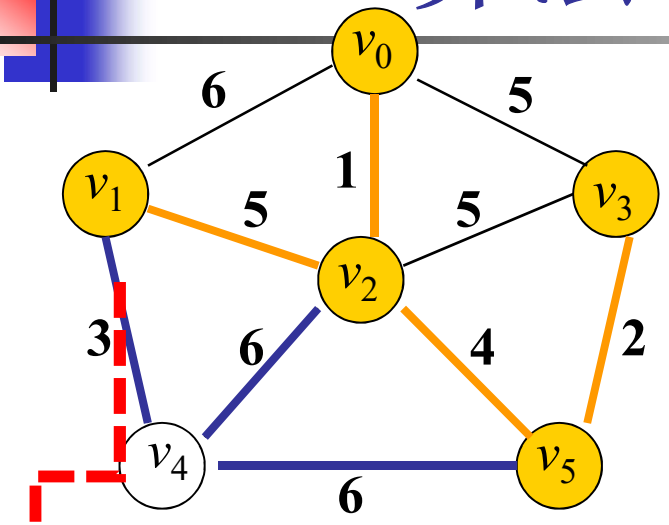
说明：蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边



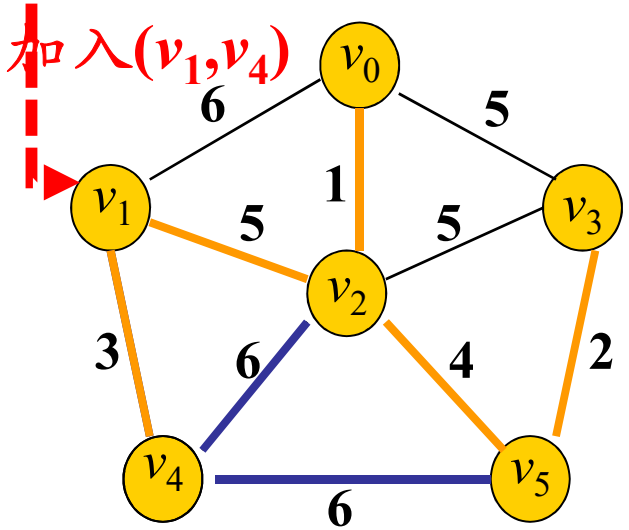
加入4条边后，生成树T=(U,TE)，U={v_0, v_2,v_5 ,v_3 ,v_1}，
TE={(v_0,v_2), (v_2,v_5), (v_5,v_3), (v_1,v_2)}

Prim算法求解过程演示

Prim算法



加入(v_1,v_4)



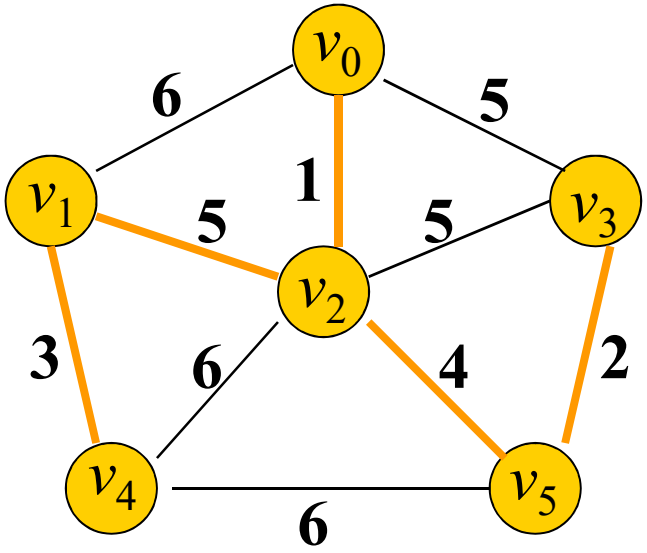
第五步：从（左侧上图）蓝色边中选一条权值最小的边(v_1,v_4)加入生成树（左侧下图）

说明：蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边；n个顶点的图做n-1步，算法停止得到最小生成树

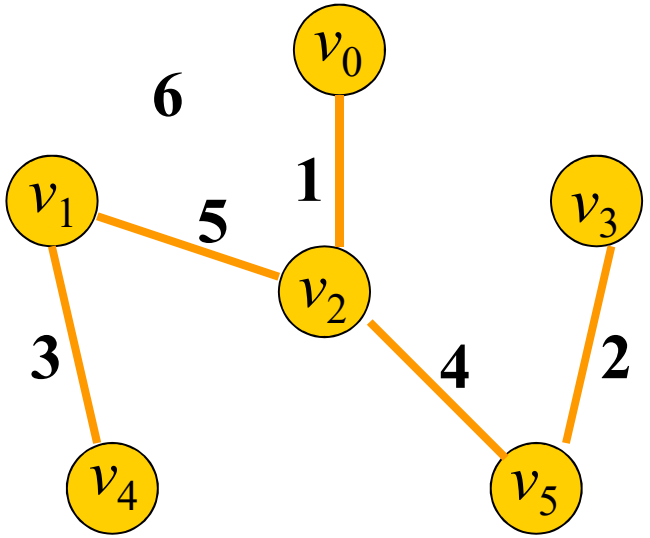
加入5条边后，生成树T=(U,TE)，$U=\{v_0, v_2, v_5, v_3, v_1, v_4\}$，$TE=\{(v_0,v_2), (v_2,v_5), (v_5,v_3), (v_1,v_2), (v_1,v_4)\}$

Prim算法求解过程演示

Prim算法

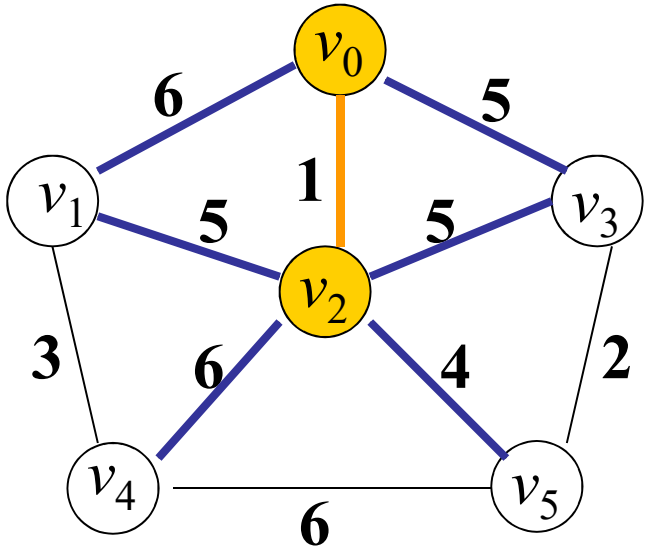


带权的连通图



Prim算法构造的最小生成树

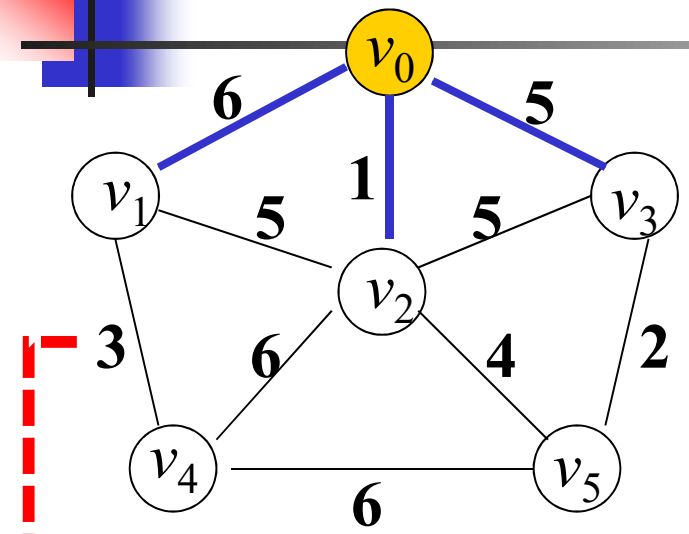
Prim算法



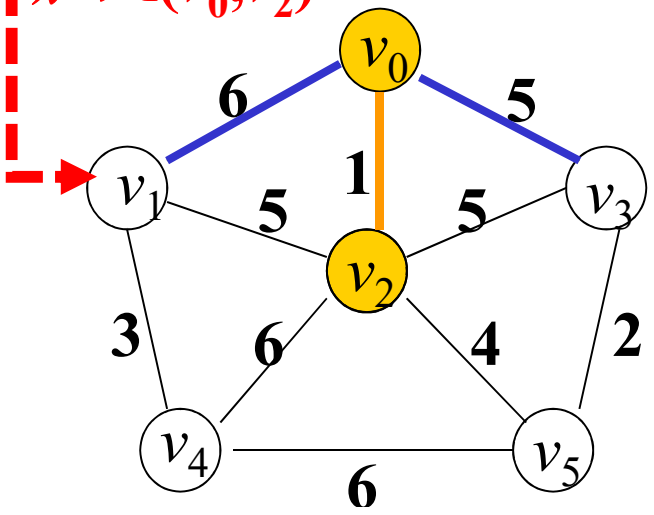
算法分析：每一步在蓝边中选一条权值最小的加入生成树中，蓝色边是一个顶点在生成树中，一个顶点不在生成树中的边；即每次要选一个不在生成树中的点加入树中。

示例中第二步蓝边6条，其中(v_1,v_0)和(v_1,v_2)都是不在生成树中的点v_1和生成树中的点相连的情况，这2条边我们根据算法只要考虑最小的那条(v_1,v_2)即可。因此实现算法时，在每一步我们只保留不在生成树中的点和生成树相连的最好情况，而不是考察不在生成树中的点和生成树相连的所有情况。

每次加入一个顶点和一条边进入生成树后，我们都考察一下不在生成树中的点和生成树中的点相连的最好情况是否被新加入的点更新



加入(v_0,v_2)



初始：任选一个顶点出发构造最小生成树，假设选v_0为出发点，将v_0加入当前的最小生成树T，当前T中只有一个顶点，0条边，即T=(U,TE)，U={v_0}，TE={}。（见左侧上图）

之后重复做n-1步，每步选一个顶点一条边加入生成树。

第一步：从蓝色边中选一条权值最小的边(v_0,v_2)加入生成树（见左侧下图）

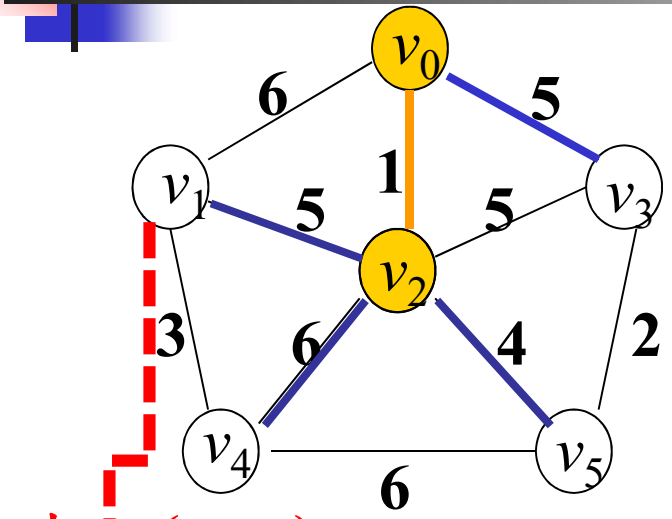
说明：蓝色边是不在生成树中的顶点与当前生成树相连的最好情况

Prim算法求解过程演示

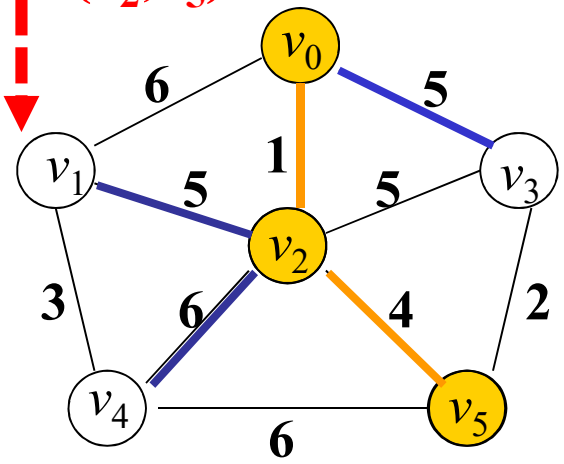
每一步保存不在生成树中的点和生成树相连的最好情况

第二步：从（左侧上图）蓝色边中选一条权值最小的边(v_2,v_5)加入生成树（左侧下图）

说明：蓝色边是不在生成树中的顶点与当前生成树相连的最好情况



加入(v_2,v_5)



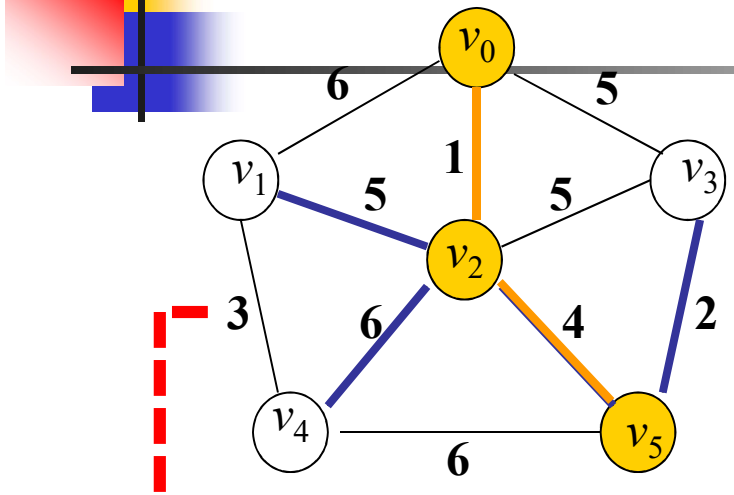
加入2条边后，生成树

T=(U,TE)，U={v_0, v_2,v_5}，

TE={(v_0,v_2)，(v_2,v_5)}

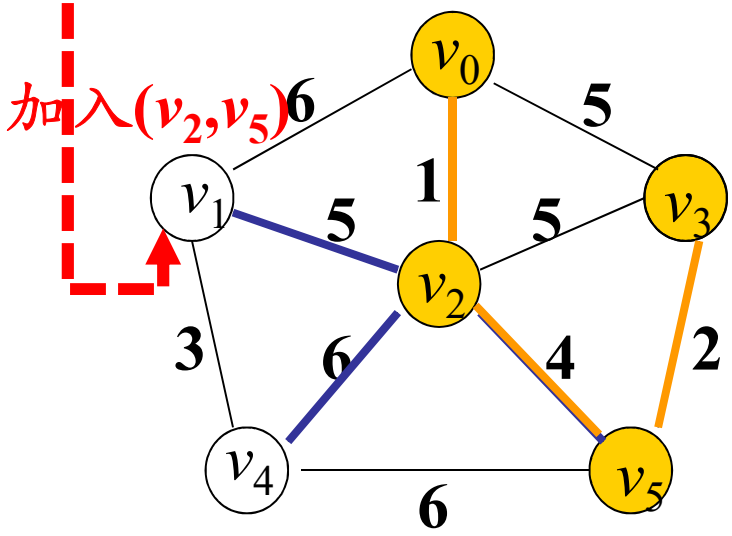
Prim算法求解过程演示

每一步保存不在生成树中的点和生成树相连的最好情况



第三步：从（左侧上图）蓝色边中选一条权值最小的边(v_5,v_3)加入生成树（左侧下图）

说明：蓝色边是不在生成树中的顶点与当前生成树相连的最好情况



加入3条边后，生成树

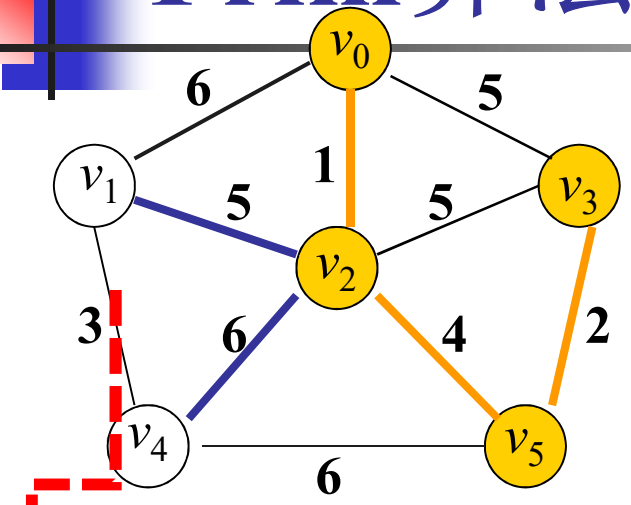
T=(U,TE)，U={v_0, v_2,v_5 ,v_3}，

TE={(v_0,v_2), (v_2,v_5), (v_5,v_3)}

Prim算法求解过程演示

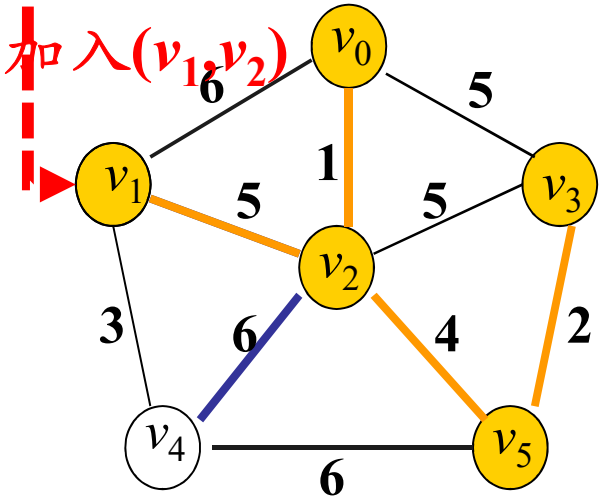
每一步保存不在生成树中的点和生成树相连的最好情况

Prim算法



第四步：从（左侧上图）蓝色边中选一条权值最小的边(v_1,v_2)加入生成树（左侧下图）

说明：蓝色边是不在生成树中的顶点与当前生成树相连的最好情况

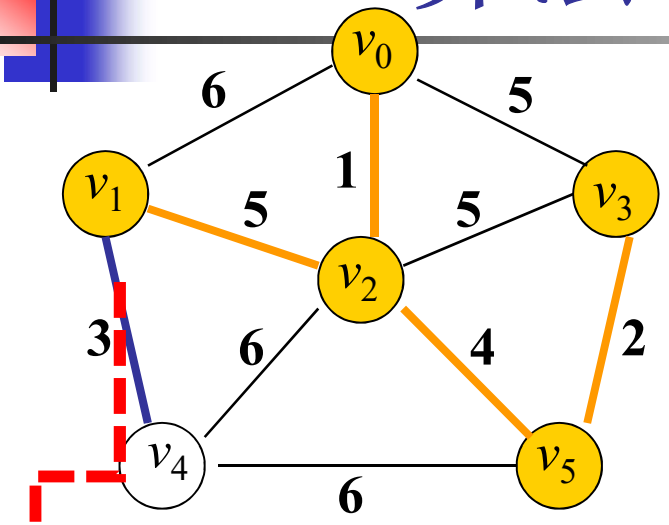


加入4条边后，生成树T=(U,TE)，$U=\{v_0, v_2, v_5, v_3, v_1\}$，$TE=\{(v_0,v_2), (v_2,v_5), (v_5,v_3), (v_1,v_2)\}$

Prim算法求解过程演示

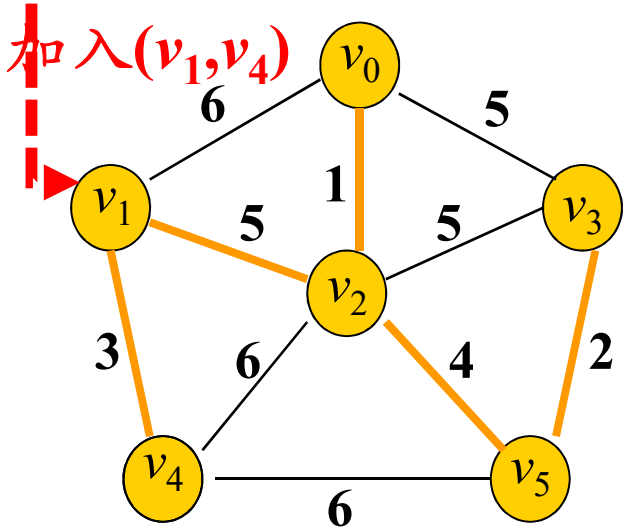
每一步保存不在生成树中的点和生成树相连的最好情况

Prim算法



第五步：从（左侧上图）蓝色边中选一条权值最小的边(v_1,v_4)加入生成树（左侧下图）

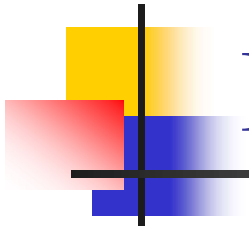
说明：蓝色边是不在生成树中的顶点与当前生成树相连的最好情况；n个顶点的图做n-1步，算法停止得到最小生成树



加入5条边后，生成树T=(U,TE)，$U=\{v_0, v_2, v_5, v_3, v_1, v_4\}$，$TE=\{(v_0,v_2), (v_2,v_5), (v_5,v_3), (v_1,v_2), (v_1,v_4)\}$

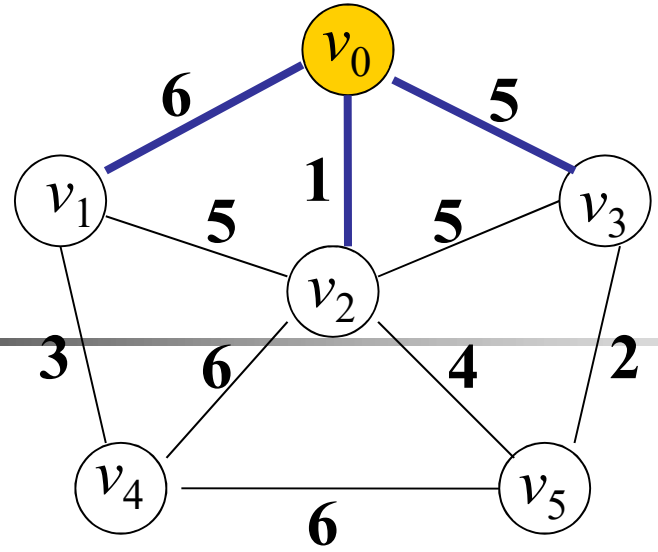
Prim算法求解过程演示

每一步保存不在生成树中的点和生成树相连的最好情况

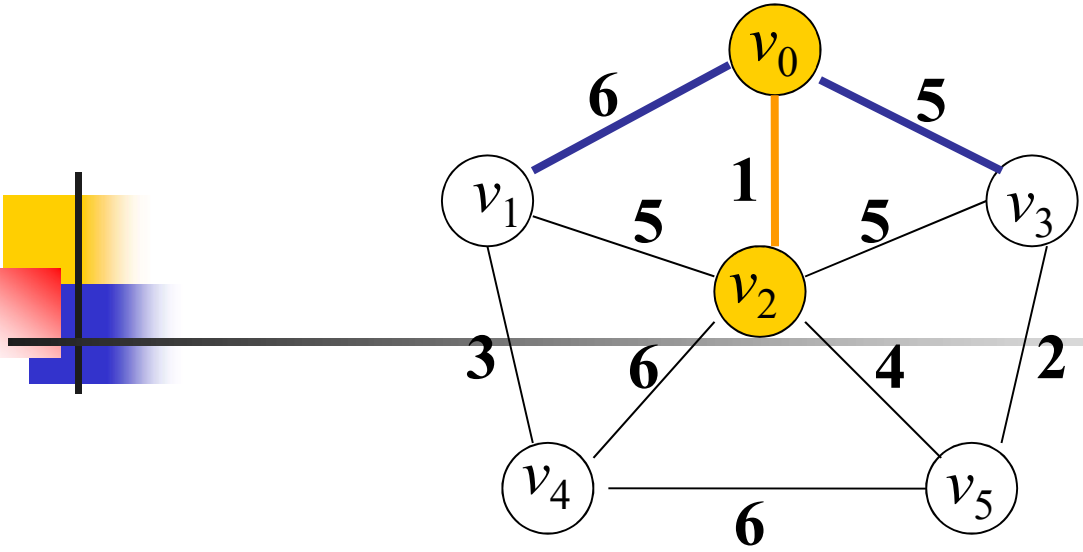


Prim算法

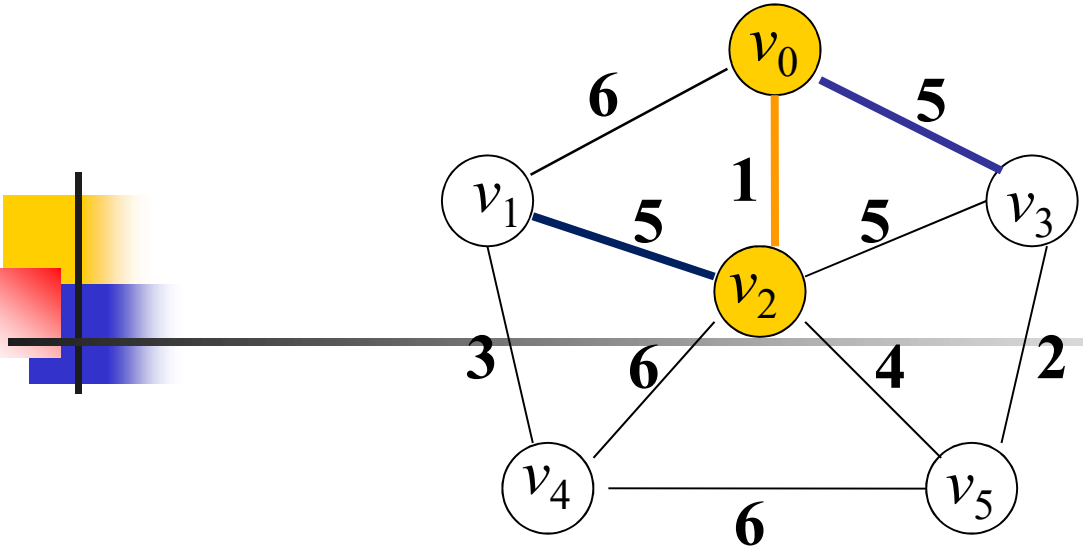
- 采用一维数组closedge[MAX]存放图中每个顶点与生成树中顶点相连的**最好**情况：
- typedef struct{
  int adjvex;//与生成树中哪个顶点相连
  int lowcost; //连边的权值
  }EdgeType;
- EdgeType closedge[MAX];//MAX是定义的一个常量
- 当顶点v尚未加入生成树时，closedge[v]存放的是v与生成树中的顶点相连的**最好**情况：v与生成树中的顶点的所有连边中权值最小的那条边； closedge[v].adjvex存放的这条权值最小的边的另一个顶点，closedge[v].lowcost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



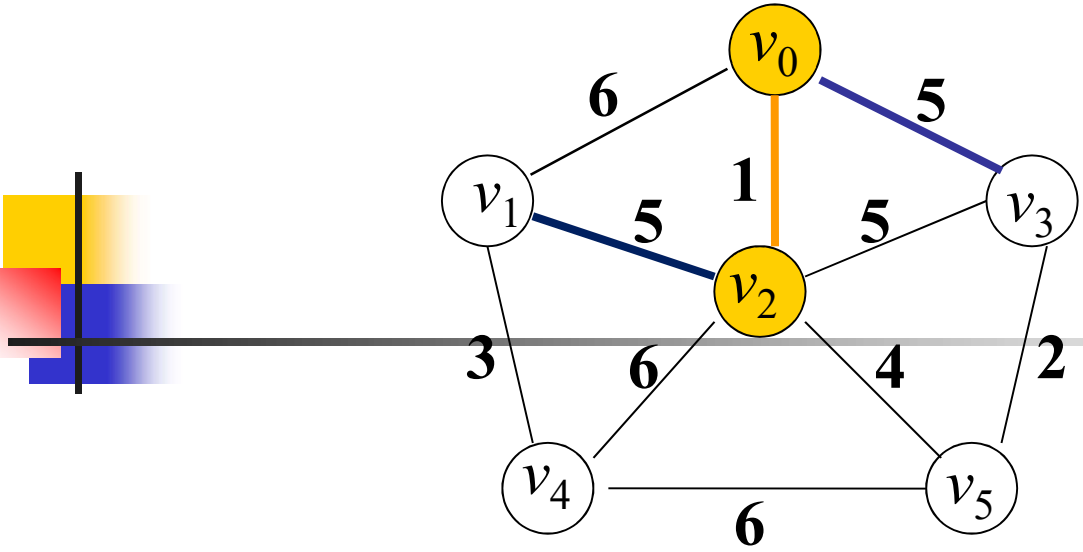
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2



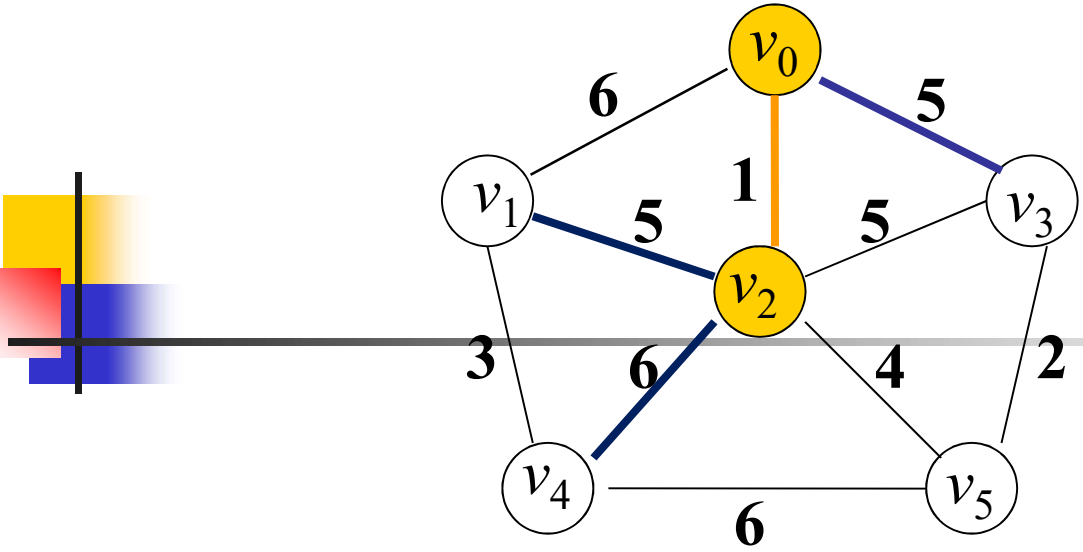
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*		v_0 1				$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



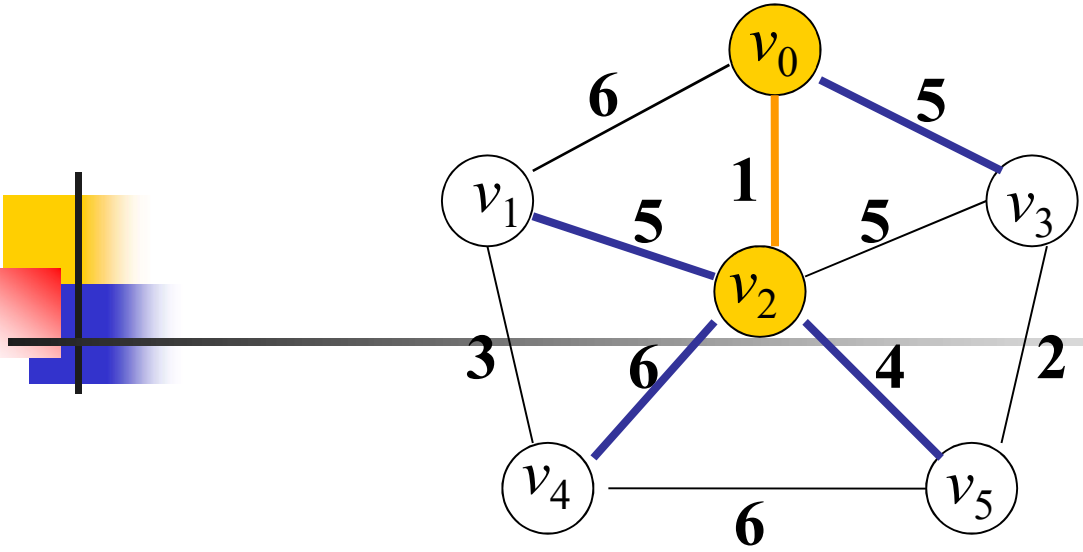
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 1				$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



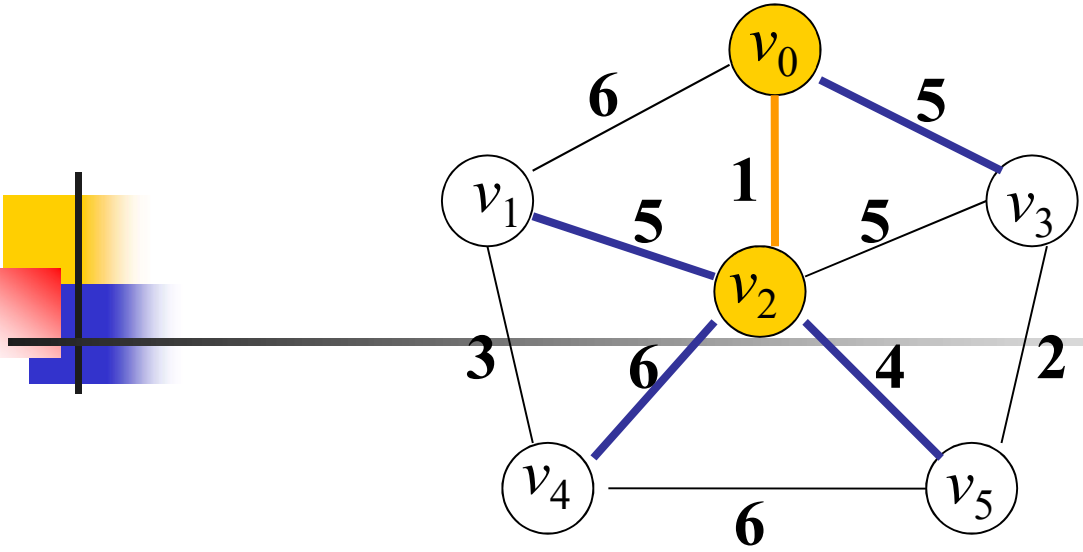
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 1	v_0 5			$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



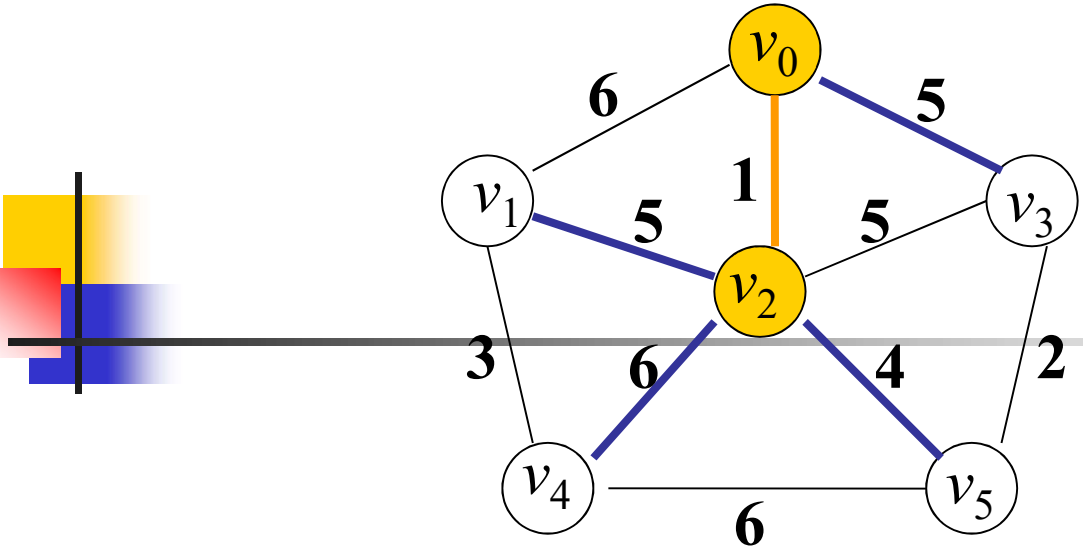
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 1	v_0 5	v_2 6		$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



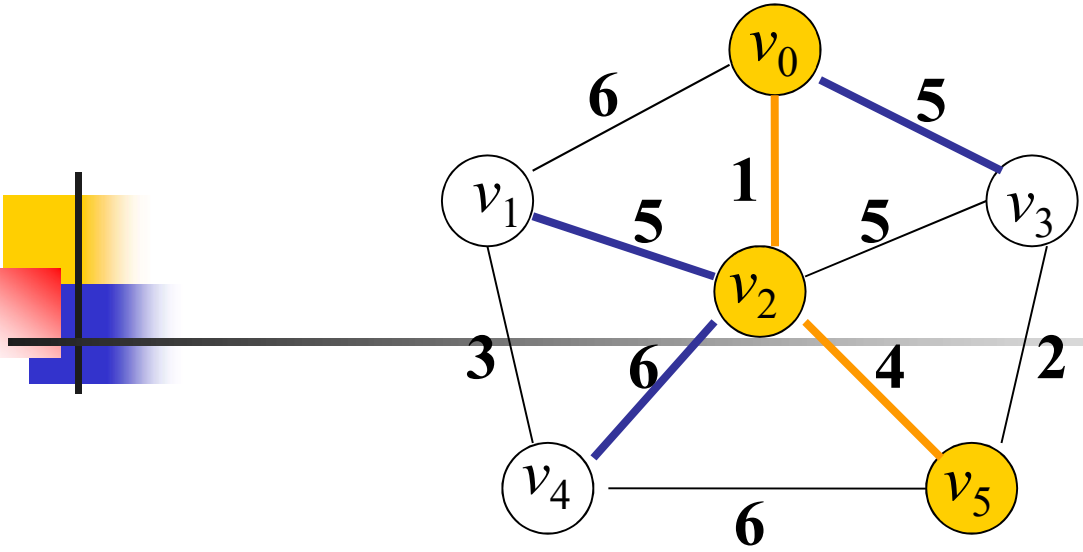
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 1	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



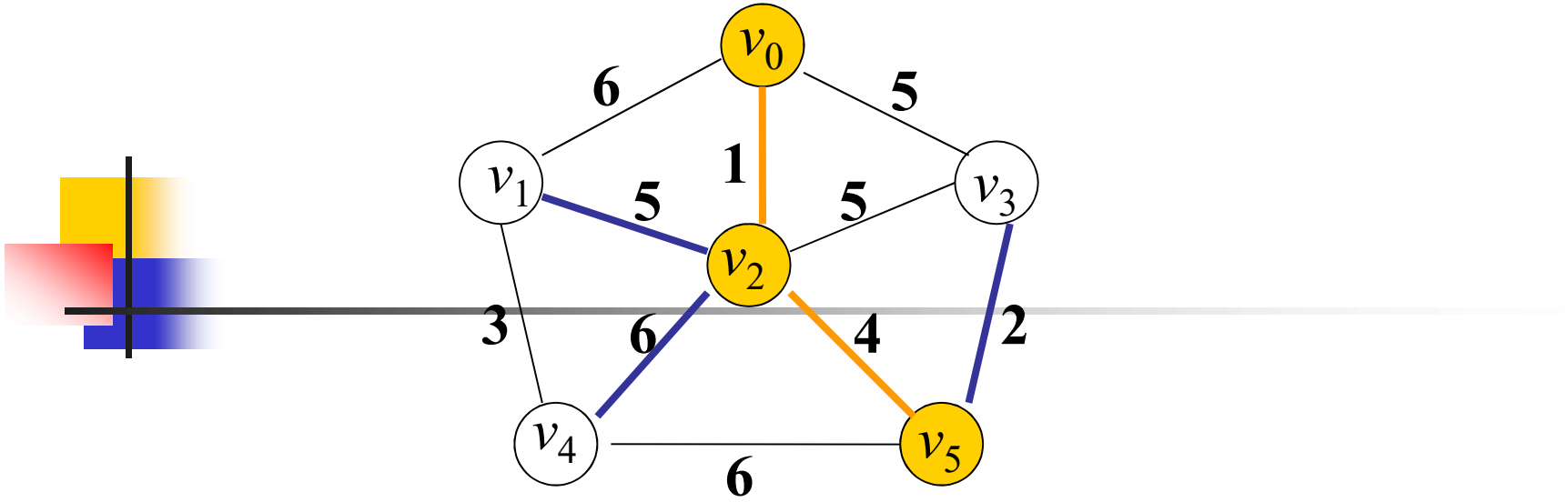
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	



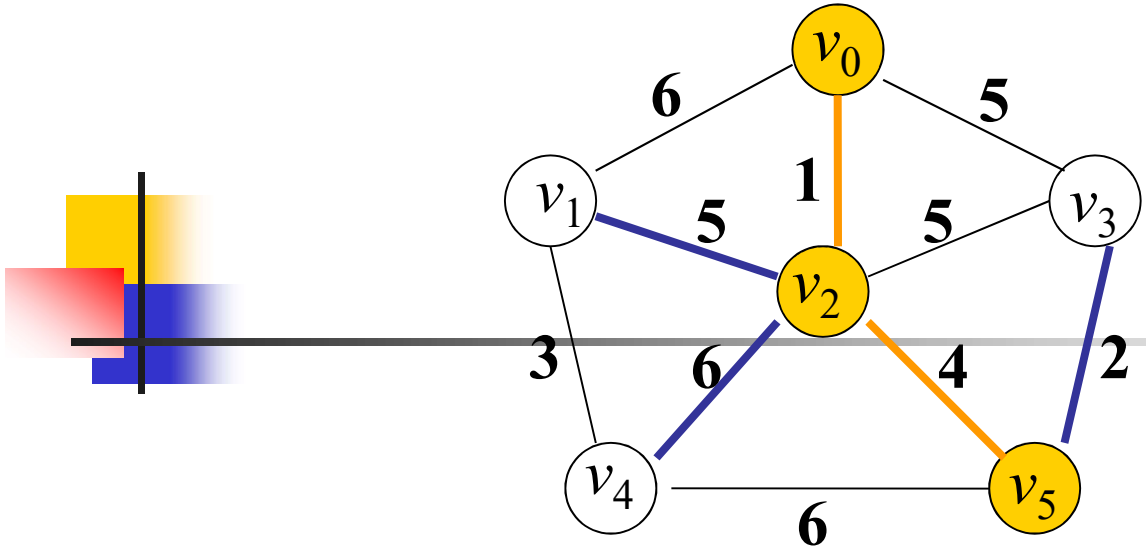
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5



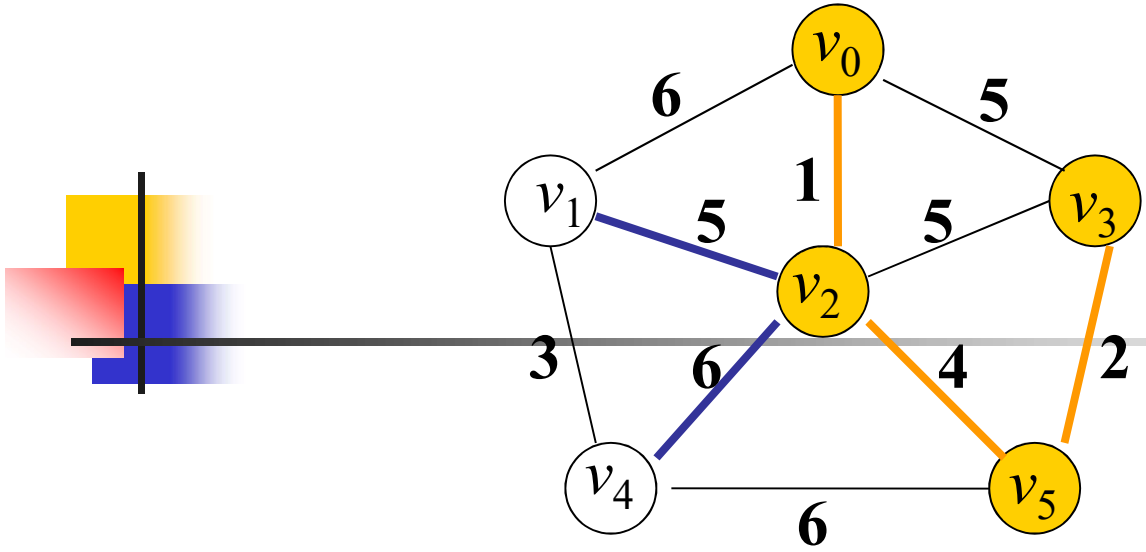
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	



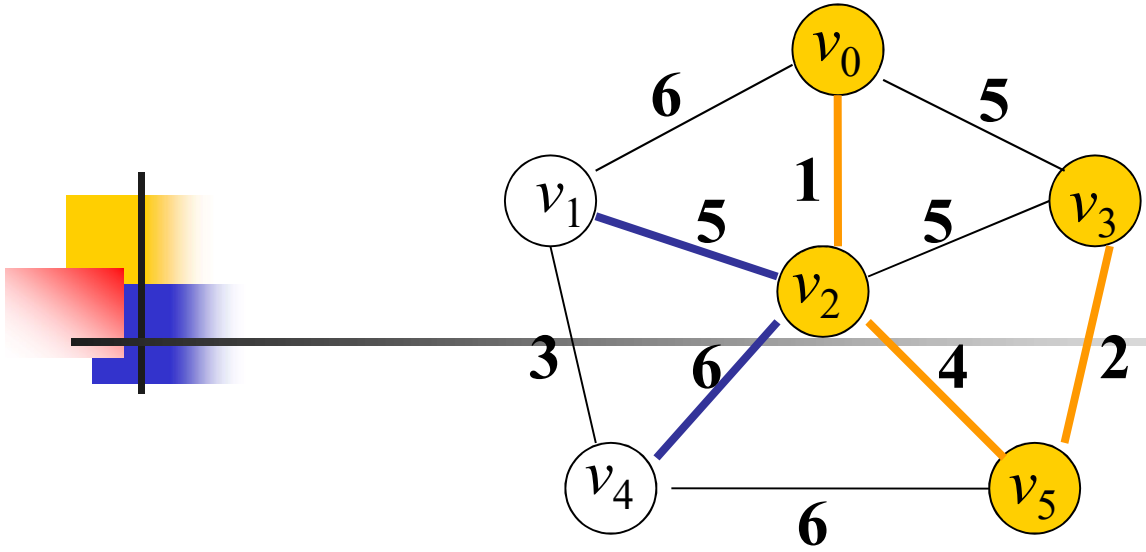
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	



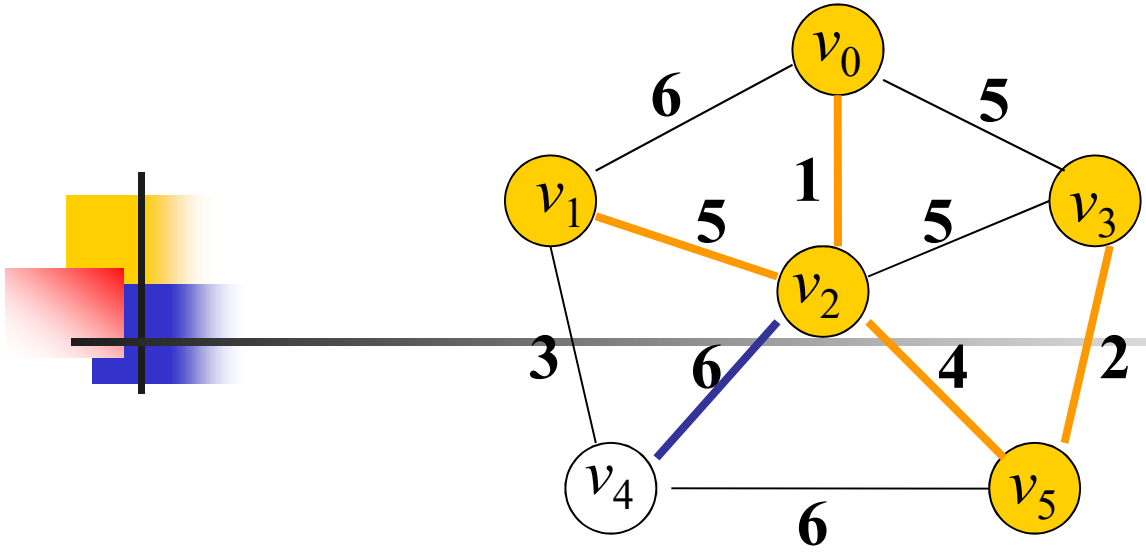
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3



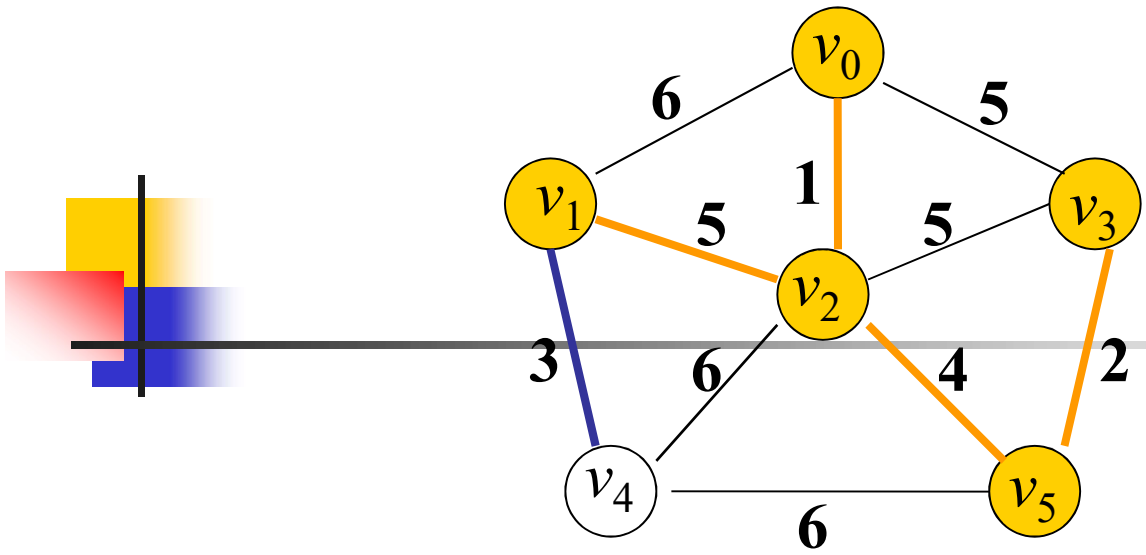
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*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



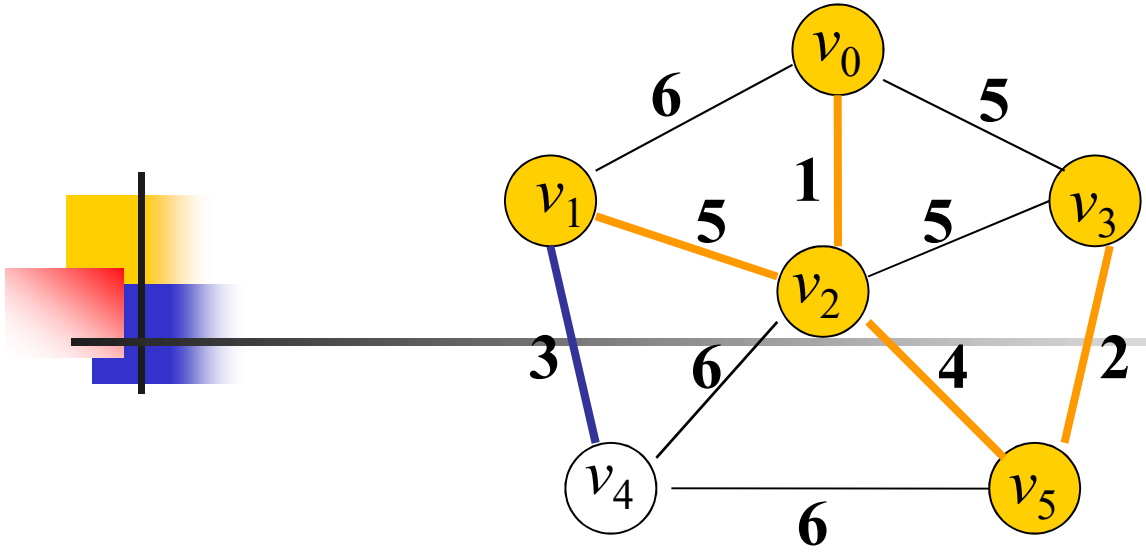
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*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



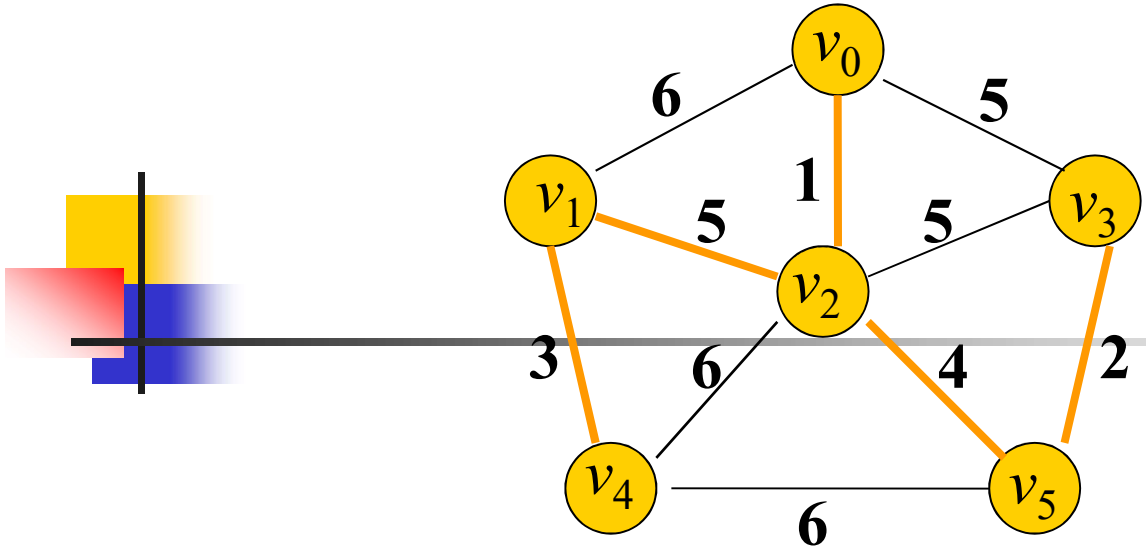
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*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



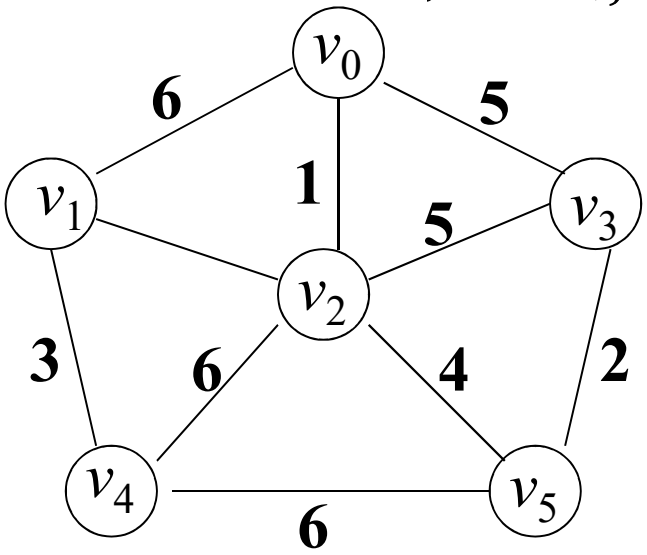
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*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



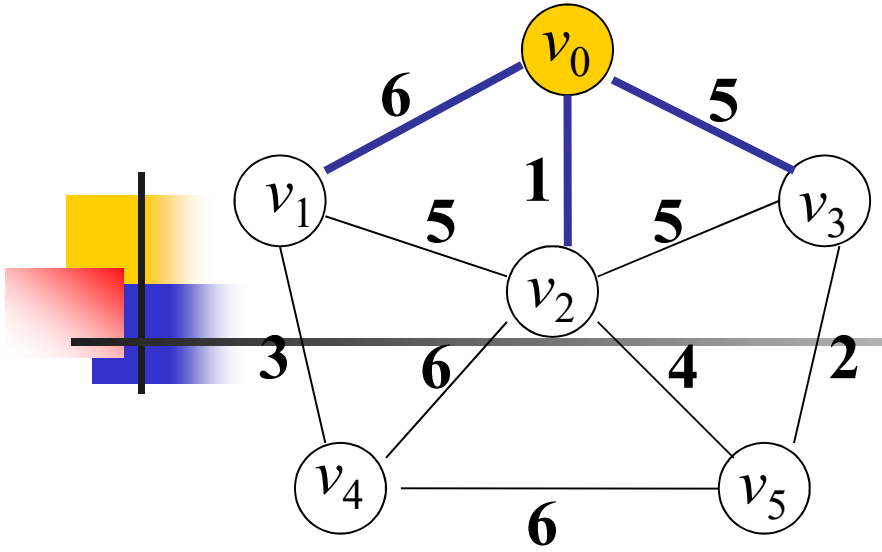
	1	2	3	4	5	U	V-U	*k*
adjvex *lowcost*	v_0 6	v_0 1	v_0 5	v_0 ∞	v_0 ∞	$\{v_0\}$	$\{v_1, v_2, v_3, v_4, v_5\}$	2
adjvex *lowcost*	v_2 5	v_0 0	v_0 5	v_2 6	v_2 4	$\{v_0, v_2\}$	$\{v_1, v_3, v_4, v_5\}$	5
adjvex *lowcost*	v_2 5	v_0 0	v_5 2	v_2 6	v_2 0	$\{v_0, v_2, v_5\}$	$\{v_1, v_3, v_4\}$	3
adjvex *lowcost*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



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



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

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

```
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：$O(n^2)$

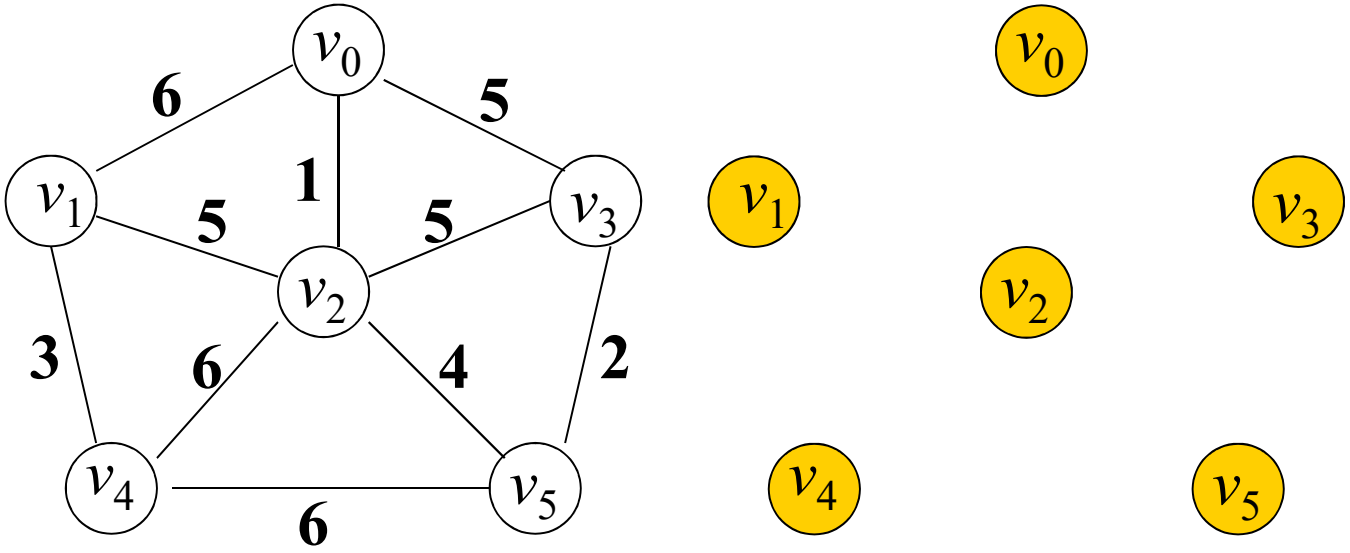
Prim算法适合于稠密图

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。

初始：所有顶点加入生成树中，当前树中没有一条边

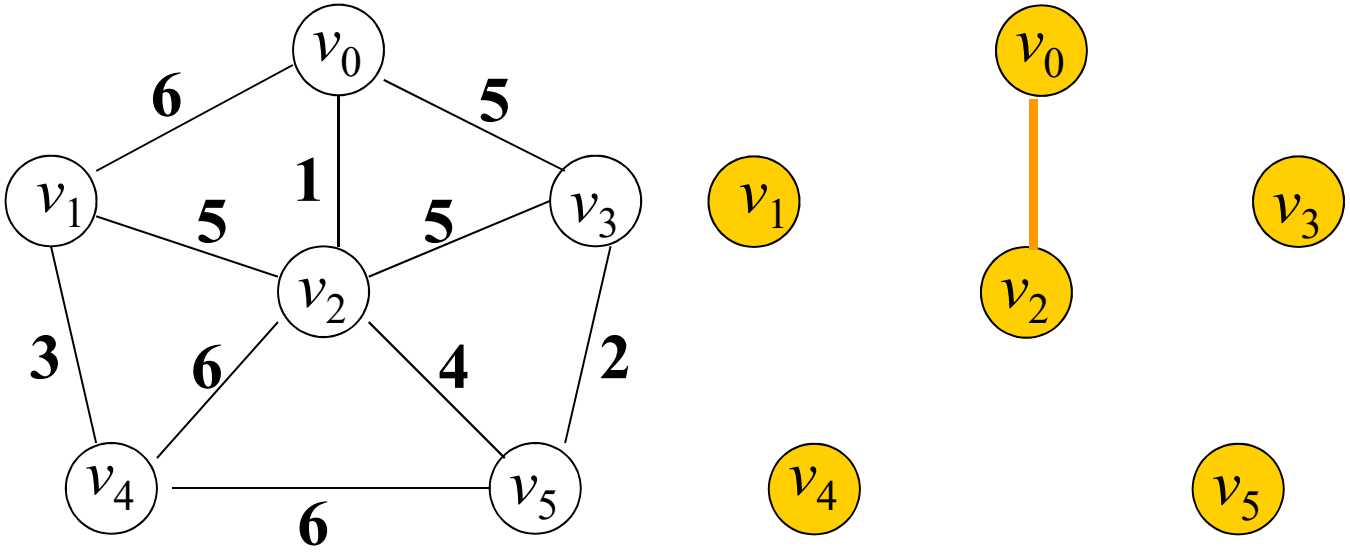
对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)



Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

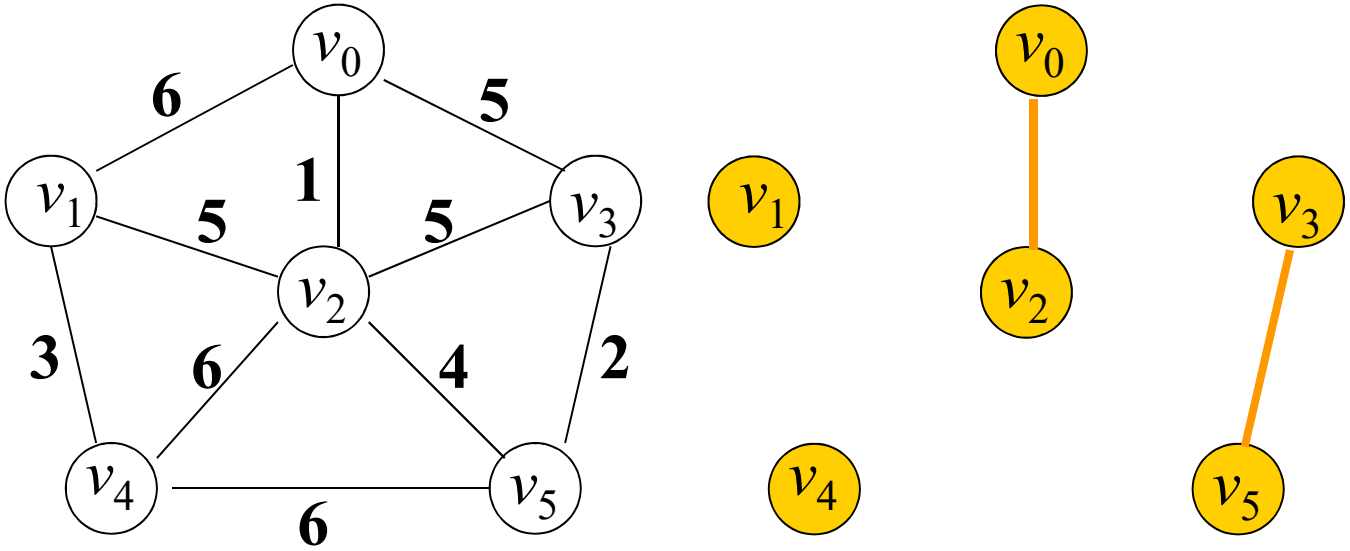
按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束



Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

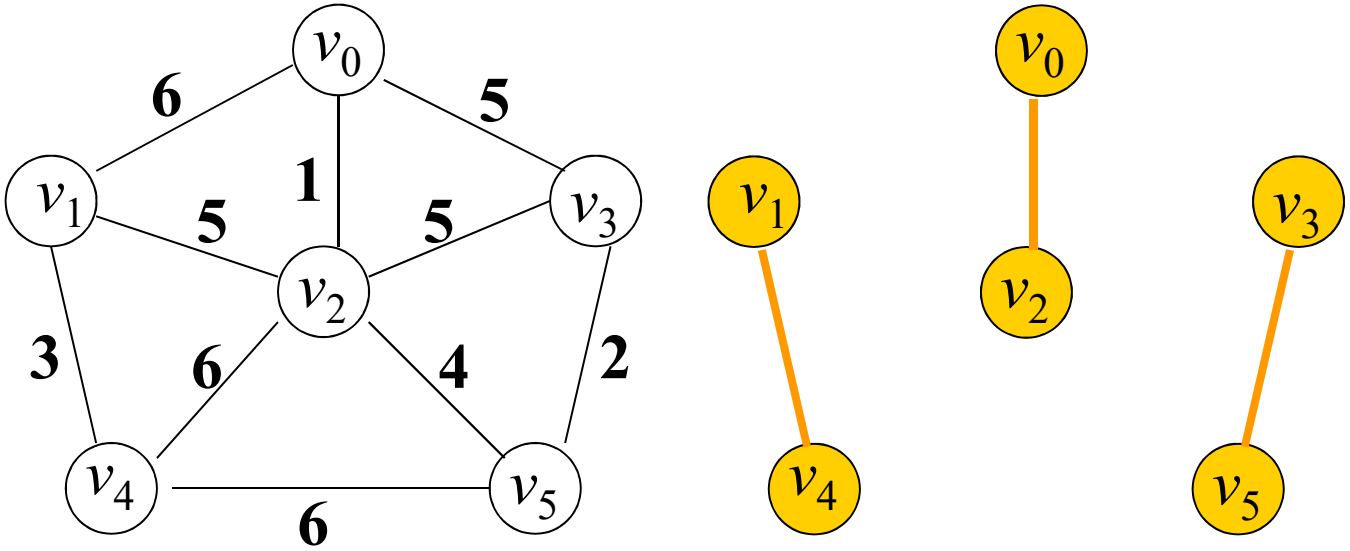
按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束



Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

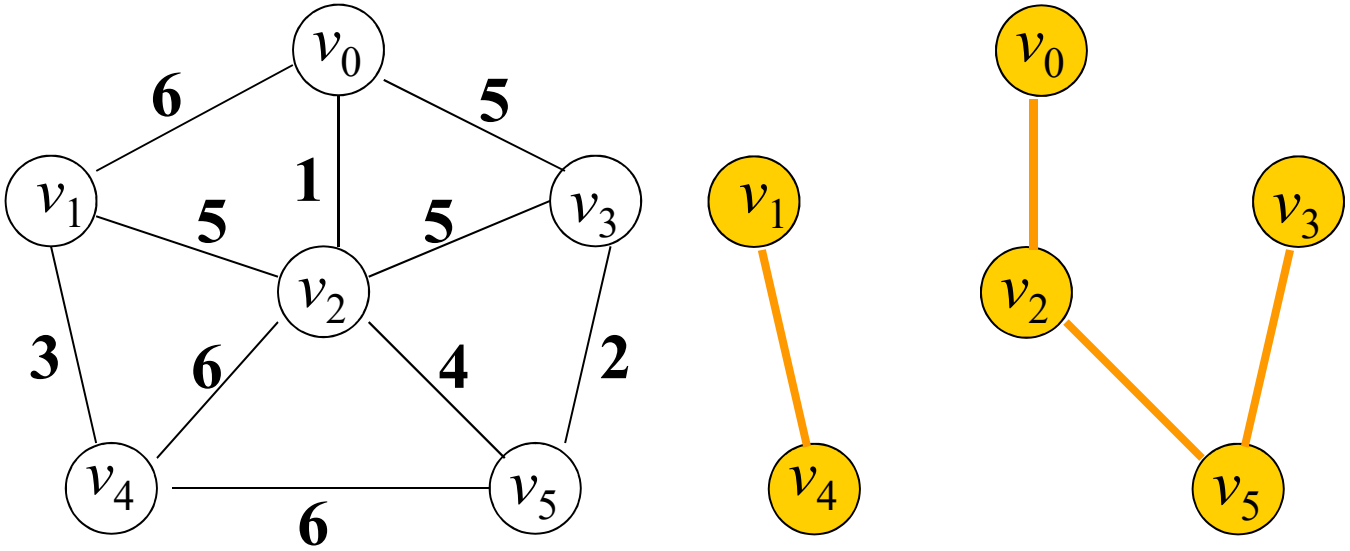
按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束



Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

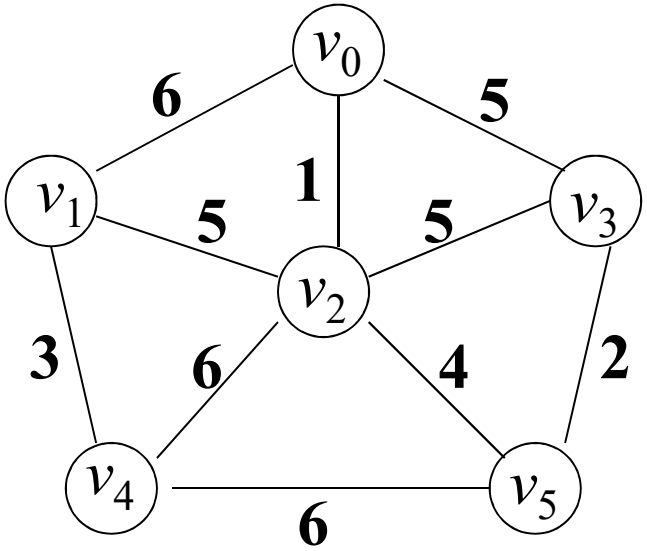
按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束

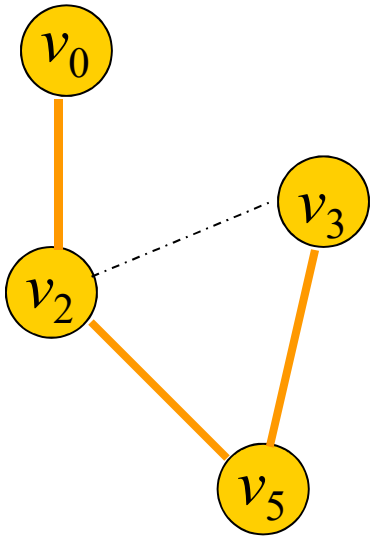


Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束



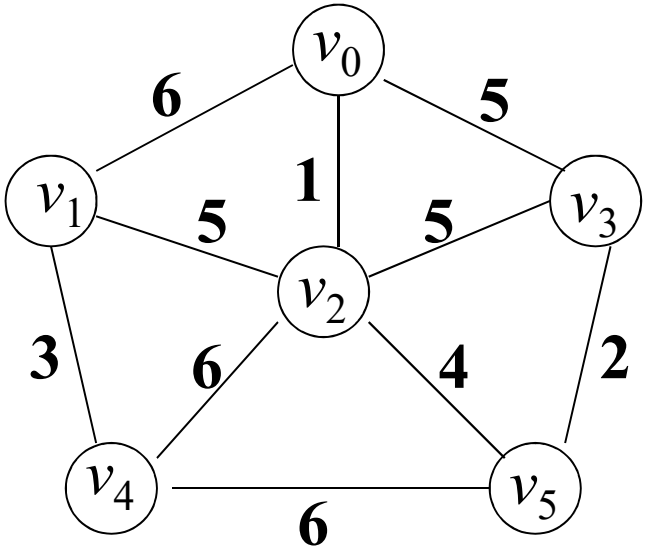


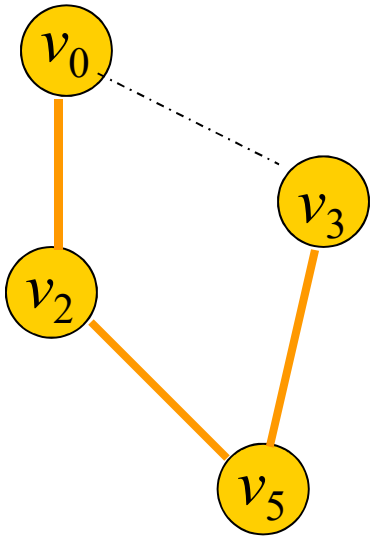
(v_2,v_3)加入当前的生成树会产生回路，所以不能加入

Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束



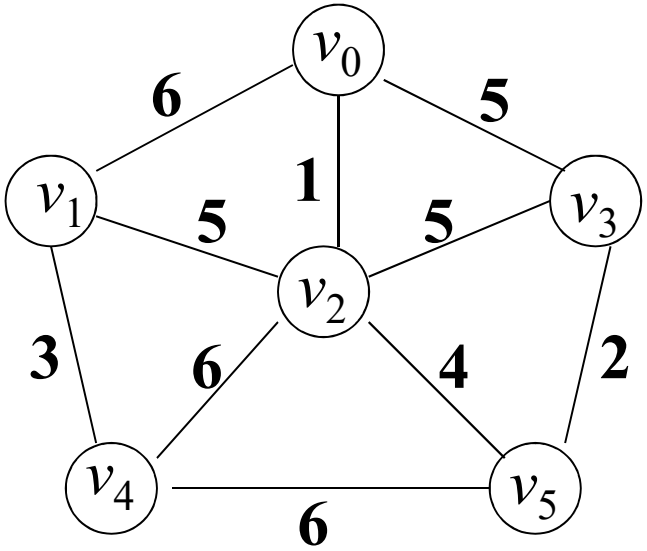


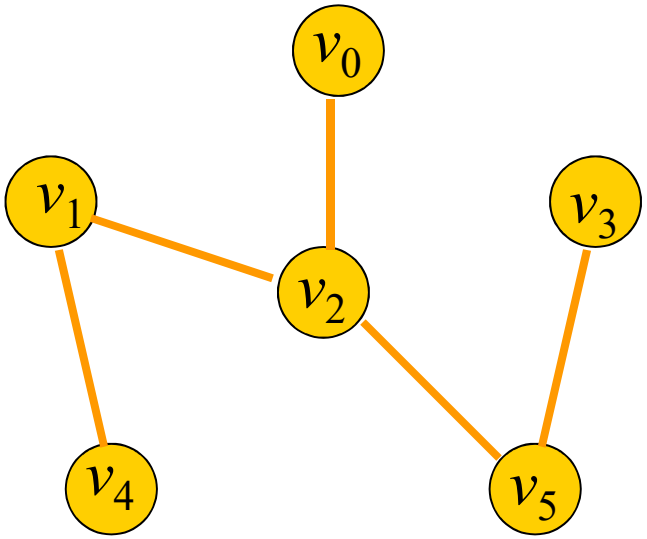
(v_0,v_3)加入当前的生成树会产生回路，所以不能加入

Kruscal算法构造最小生成树的过程演示

对所有边按照权值升序排序：(v_0,v_2), (v_3,v_5), (v_1,v_4), (v_2,v_5), (v_2,v_3), (v_0,v_3), (v_2,v_1), (v_0,v_1), (v_4,v_2), (v_5,v_4)

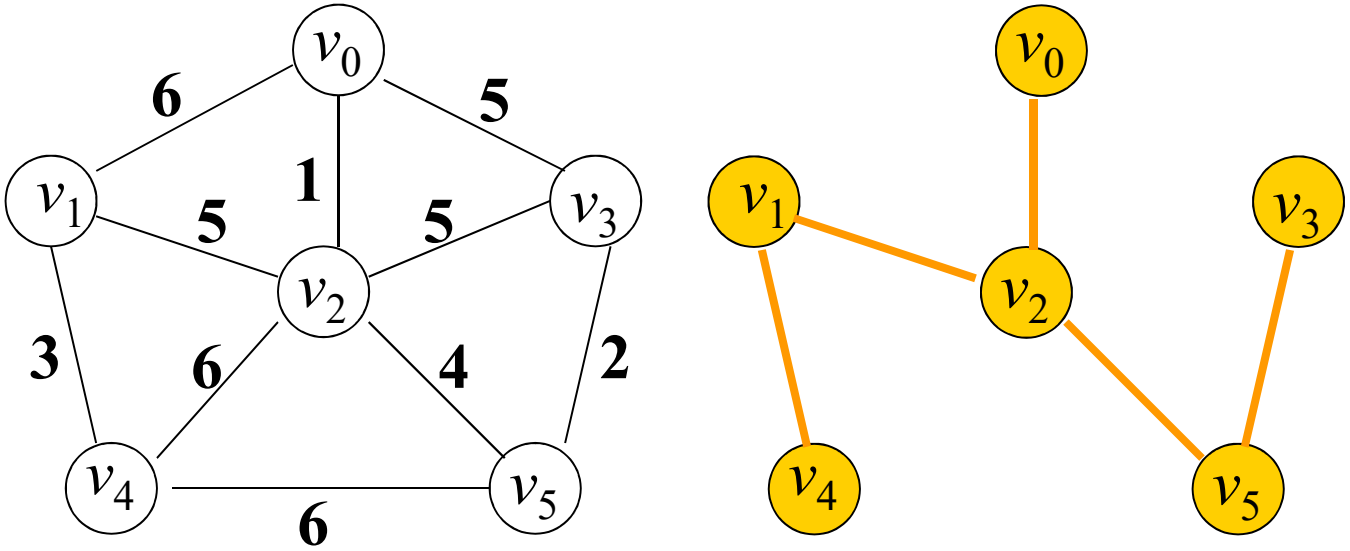
按照边的升序依次考察每一条边，直到加入n-1条边或所有边都考察结束





(v_2,v_1)加入当前的生成树不会产生回路，所以加入

Kruscal算法构造最小生成树的过程演示



Kruscal算法适合稀疏图，时间复杂度为O(eloge)，e为图的边数，因为该算法要对边按照权值排序，（堆、快速。归并）排序算法的平均时间复杂度O(eloge)