图与网络模型及方法

1. 图的基本概念与数据结构
2. 最短路问题
3. 最小生成树问题
4. 网络的最大流问题
5. 最小费用最大流问题
6. 欧拉图与中国邮差问题
7. 旅行商问题

在有限元素的离散系统中，相应的数学模型又可以划分为两类，一类是存在有效算法的所谓**P**类问题，即多项式时间内可以解决的问题。

若有一个算法（譬如图灵机，或一个**LISP**或**Pascal**的程序并有无限的内存）能够在最多n^k步内对一个串长度为**n**的输入给出正确答案，其中**k**是某个不依赖于输入串的常数，则我们称该问题可以在多项式时间内解决，并且将它置入类**P**。直观的讲，我们将**P**中的问题视为可以较快解决的问题。

但是这类问题在**MCM**中非常少见，事实上，由于竞赛是开卷的，参考相关文献，使用现成的算法解决一个**P**类问题，不能显示参赛者的建模及解决实际问题能力之大小；

还有一类所谓的NP问题，这种问题每一个都尚未建立有效的算法，也许真的就不可能有有效算法来解决。命题往往以这种NPC问题为数学背景，找一个具体的实际模型来考验参赛者。这样增加了建立数学模型的难度。但是这也并不是说无法求解。一般来说，由于问题是具体的实例，我们可以找到特殊的解法，或者可以给出一个近似解。

图论作为离散数学的一个重要分支，在工程技术、自然科学和经济管理中的许多方面都能提供有力的数学模型来解决实际问题，所以吸引了很多研究人员去研究图论中的方法和算法。

我们对图论中的经典例子或多或少还是有一些了解的，比如，哥尼斯堡七桥问题、中国邮递员问题、四色定理等等。图论方法已经成为数学模型中的重要方法。许多难题由于归结为图论问题被巧妙地解决。而且，从历年的数学建模竞赛看，出现图论模型的频率极大，比如：

CMCM93B－足球队排名(特征向量法)

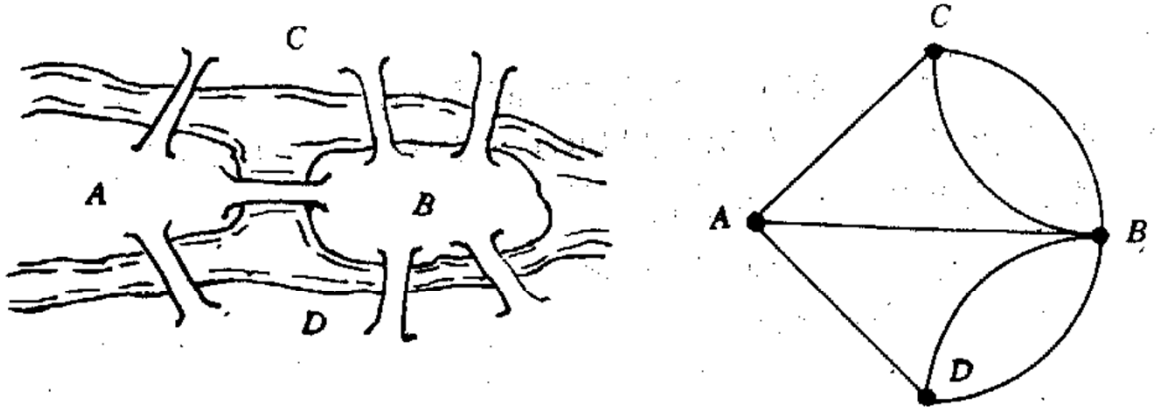
CMCM94B－锁具装箱问题(最大独立顶点集、最小覆盖等用来证明最优性)

CMCM98B－灾情巡视路线(最优回路)

这里面都直接或是间接用到图论方面的知识。要说明的是，这里图论只是解决问题的一种方法，而不是唯一的方法。

图论起源于18世纪。第一篇图论论文是瑞士数学家欧拉于1736 年发表的“哥尼斯堡的七座桥”。

在哥尼斯堡有七座桥将普莱格尔河中的两个岛及岛与河岸联结起来，问如何才能从这四块陆地中的任何一块开始通过每一座桥正好一次，再回到起点。

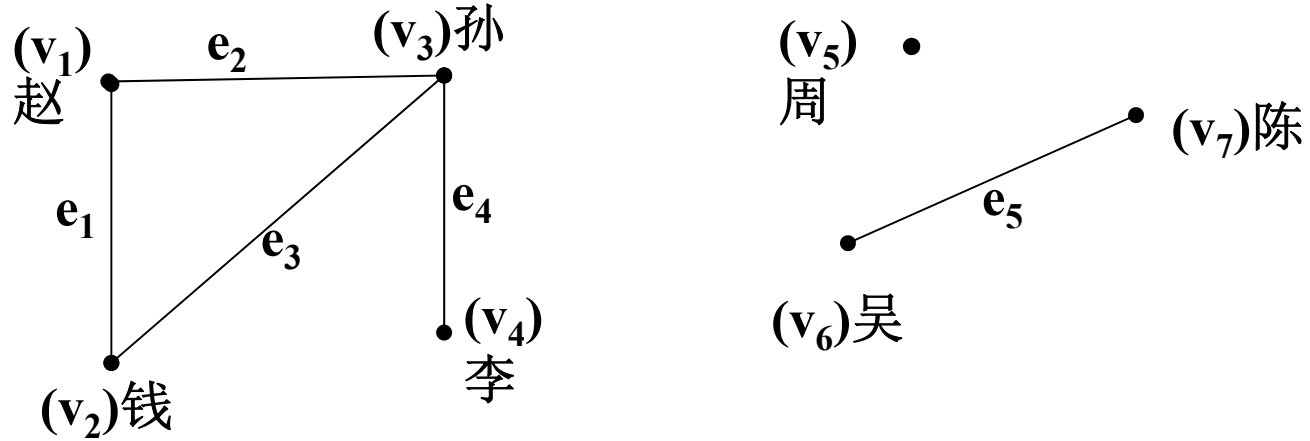


哥尼斯堡问题归结为一个图形的一笔画问题，即是否能从某一点开始，一笔画完成这个图形，最后回到原点而不重复。

1847年，克希霍夫为了给出电网络方程而引进了“树”的概念。1857年，凯莱在计数烷 的同分异构物时，也发现了“树”。哈密尔顿于1859年提出“周游世界”游戏，用图论的术语，就是如何找出一个连通图中的生成圈、近几十年来，由于计算机技术和科学的飞速发展，大大地促进了图论研究和应用，图论的理论和方法已经渗透到物理、化学、通讯科学、建筑学、运筹学、生物遗传学、心理学、经济学、社会学等学科中。

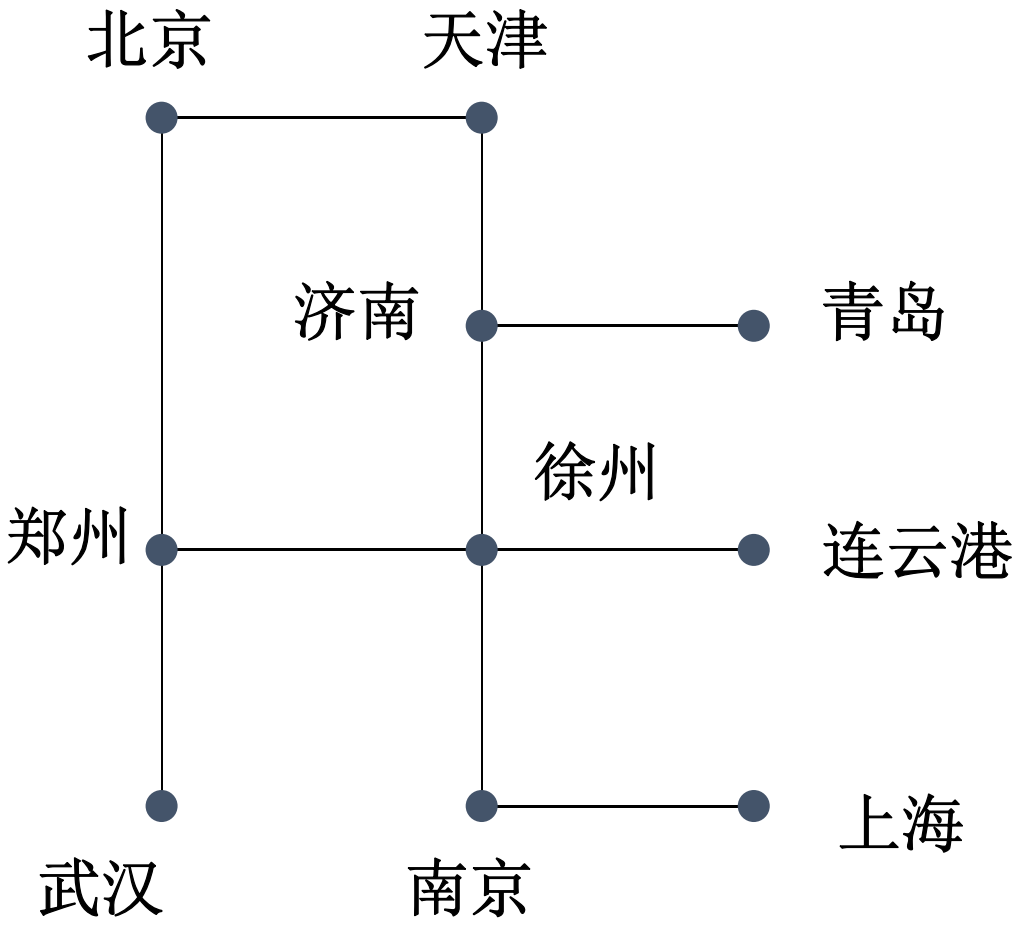
图论中所谓的“图”是指某类具体事物和这些事物之间的联系。如果我们用点表示这些具体事物，用连接两点的线段（直的或曲的）表示两个事物的特定的联系，就得到了描述这个“图”的几何形象。图论为任何一个包含了一种二元关系的离散系统提供了一个数学模型，借助于图论的概念、理论和方法，可以对该模型求解。

例如：在一个人群中，对相互认识这个关系我们可以用图来表示，下图就是一个表示这种关系的图。



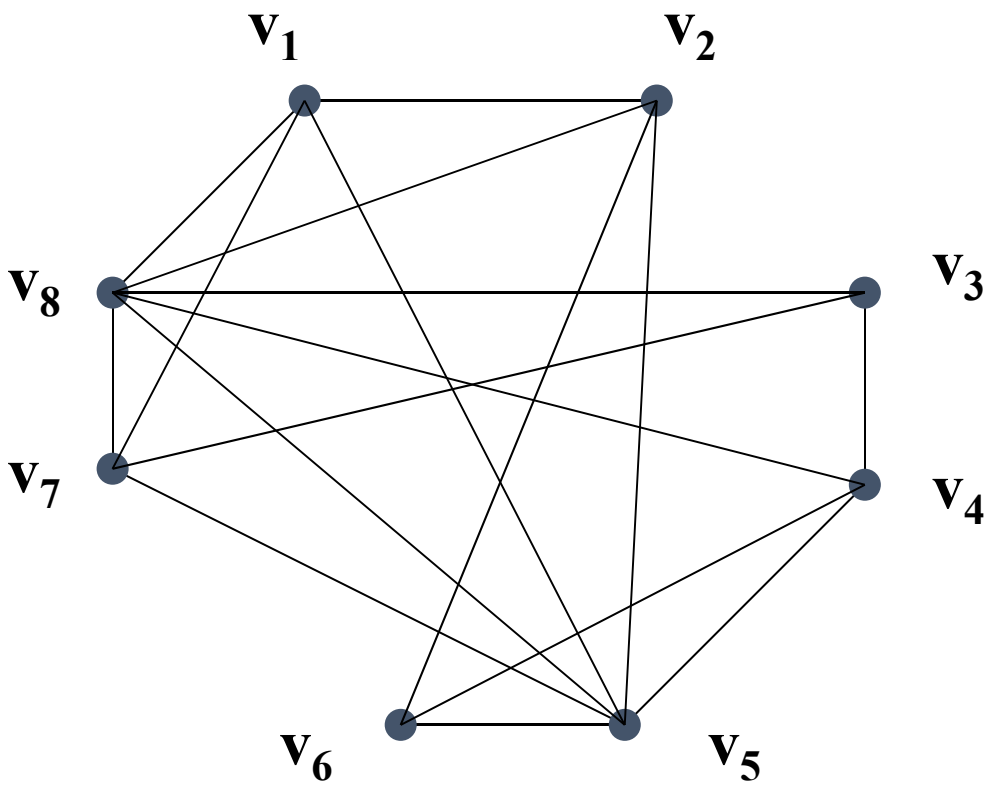
- 图与网络用来反映一些对象之间的关系
- 图论中，图与网络结构由下面两个元素构成
  - 节点
  - 边

又比如右图是北京、上海等十个城市间的铁路交通图。与此类似的还有电话线分布图、煤气管道图、航空路线图等。



“染色问题”

储存8种化学药品，其中某些药品不能存放在同一个库房里。 用$v_1,v_2,\ldots,v_8$分别代表这8种药品。规定若两种药品不能存放在一起，则其相应的点之间联一条线。如下图所示：



可知需要4个库房，其中一个答案是：

$\{ v_1 \}$

$\{ v_2, v_4, v_7 \}$

$\{ v_3, v_5 \}$

$\{ v_6, v_8 \}$

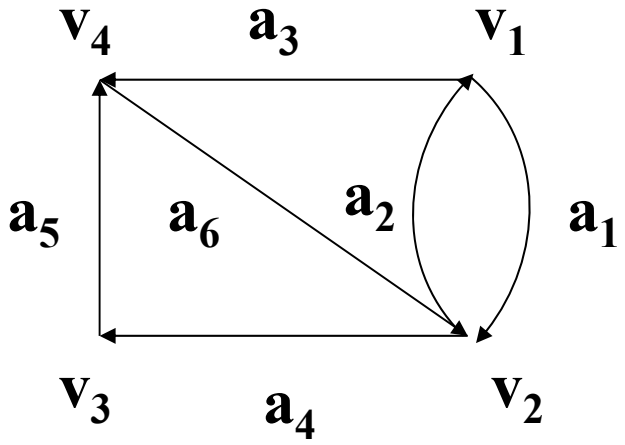
还有其他的答案。

1. 图的基本概念与定理

1.1图的基本概念

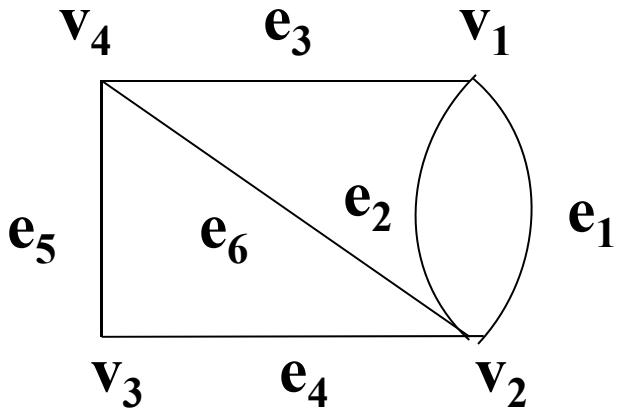
- 有向图

由点及弧所构成的图，记为D=(V,A)， V,A分别是D的点集合和弧集合。



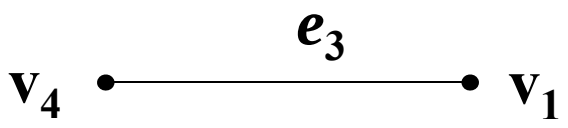
- 无向图

由点及边所构成的图。记为G=(V,E)， V，E分别是G的点集合和边集合。



- 边

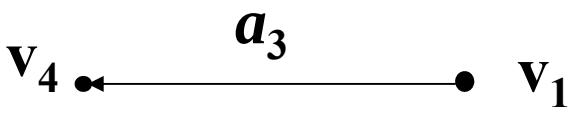
两点之间不带箭头的联线。



如 e_3

- 弧

两点之间带箭头的联线。

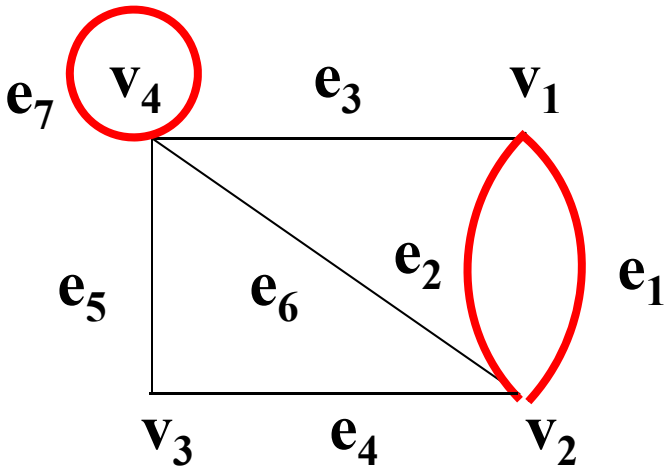


如 a_3

- 端点及关联边

若边$e=[u,v]\in E$，则称u,v为e的端点，也称u,v是相邻的，称e是点u（及点v）的关联边。

如：v_1,v_4为e_3的端点，v_1,v_4是相邻的，e_3是v_1（v_4）的关联边。



- 环

若在图G中，某个边的两个端点相同，则称e是环。如 e_7

- 多重边

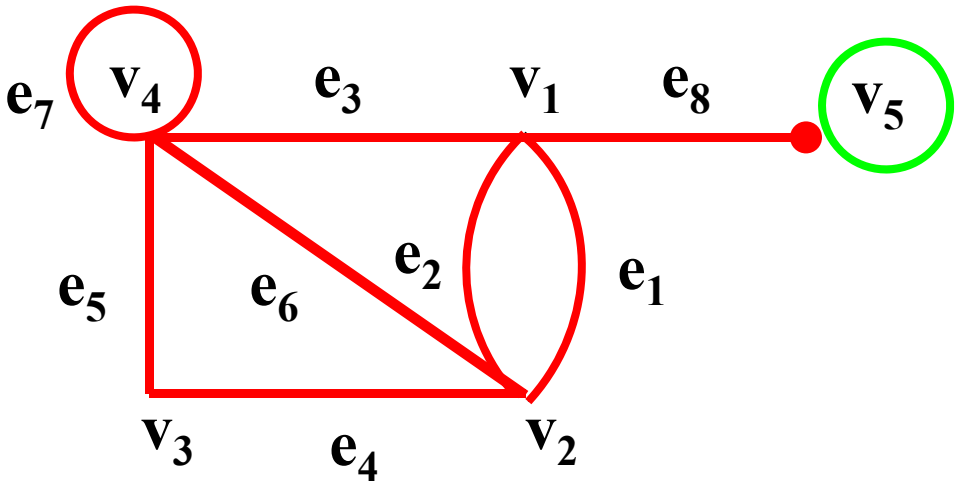
若两个点之间有多于一条的边，称这些边为多重边。如 e_1,e_2

- 简单图

一个无环，无多重边的图。

- 多重图

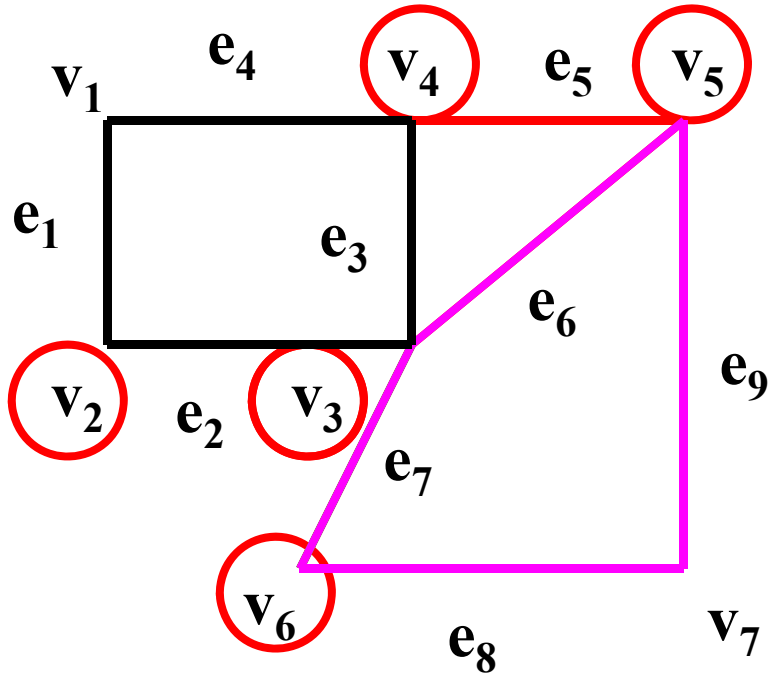
一个无环、但允许有多重边的图。



- 点v的次　　如 $d(v_4)=5$

  以点v_i为端点边的个数，记为$d_G(v_i)$或$d(v_i)$。　　$d(v_2)=4$

- 悬挂点　次为1的点，如 v_5
- 悬挂边　悬挂点的关联边，如 e_8
- 孤立点　次为0的点
- 偶点　次为偶数的点，如 v_2
- 奇点　次为奇数的点, 如 v_5



- 链
- 中间点
- 初等链
- 圈
- 初等圈
- 简单圈

在上图中,$(v_1,v_2,v_3,v_4,v_5,v_3,v_6,v_7)$是一条链, 但不是初等链

在该链中,v_2,v_3,v_4,v_5,v_3,v_6是中间点

(v_1,v_2,v_3,v_6,v_7)是一条初等链

$(v_4,v_1,v_2,v_3,v_5,v_7,v_6,v_3,v_4)$是一个简单圈

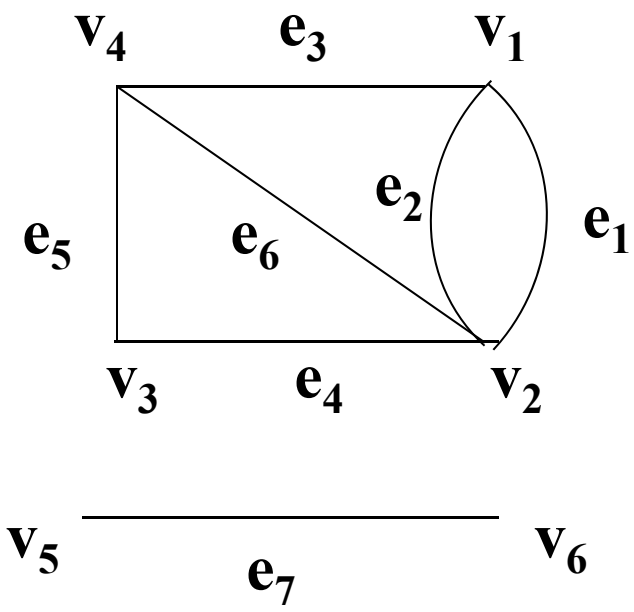
(v_1,v_2,v_3,v_4,v_1)是一个初等圈

- 连通图

图G中，若任何两个点之间，至少有一条链，称为连通图。否则称为不连通图。

- 连通分图(分图)

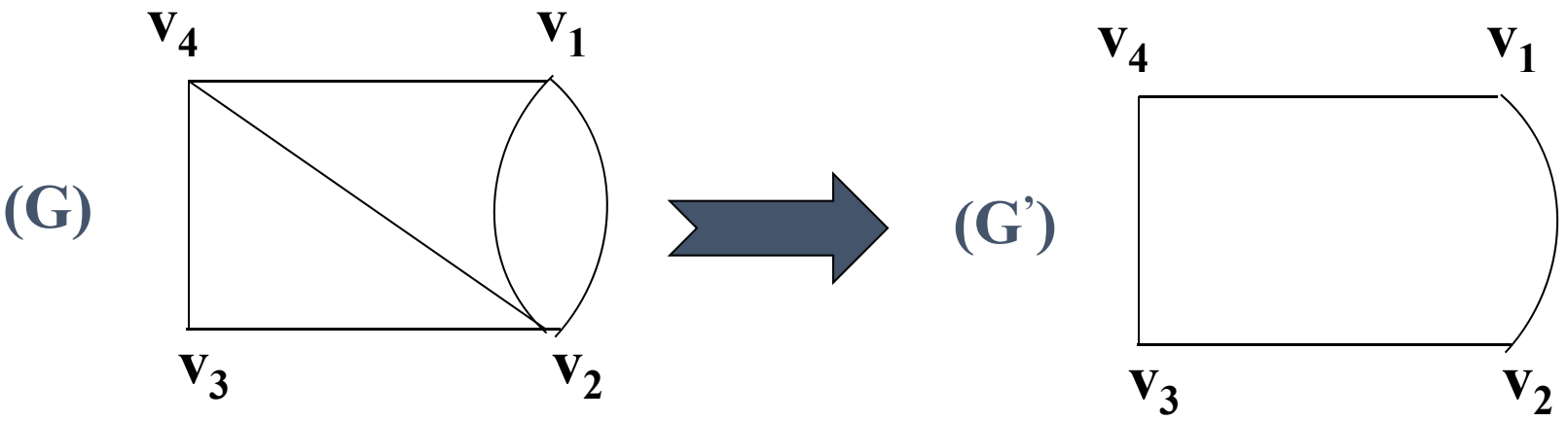
若G是不连通图，则它的每个连通的部分称为连通分图。



如左图就是个不连通图，它是由两个连通分图构成的。

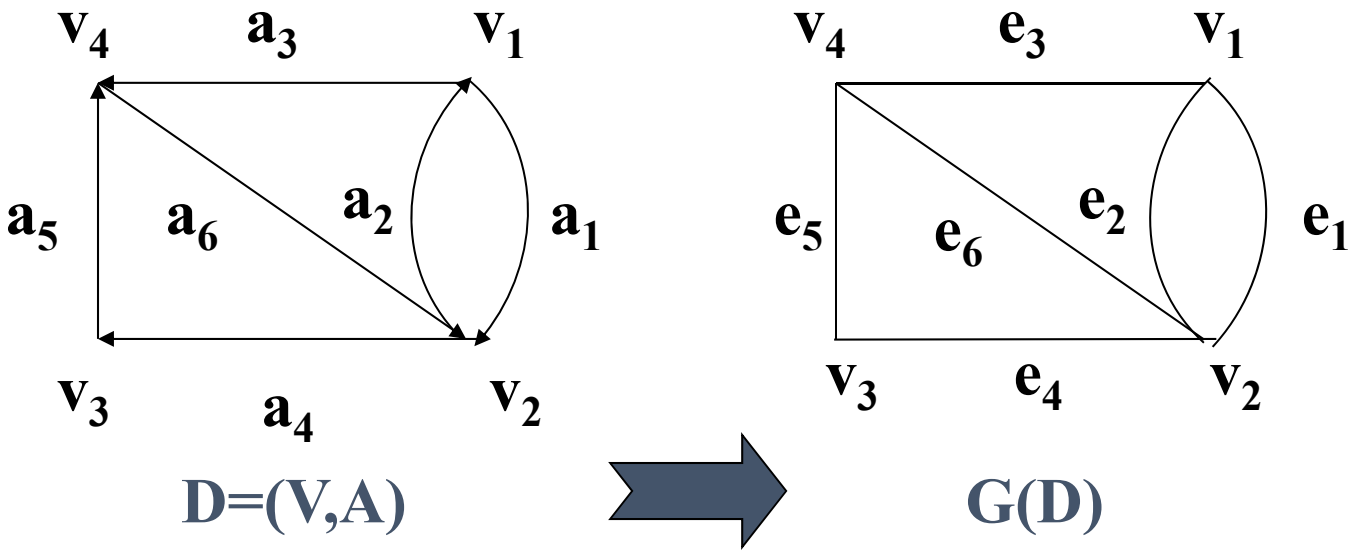
- 支撑子图

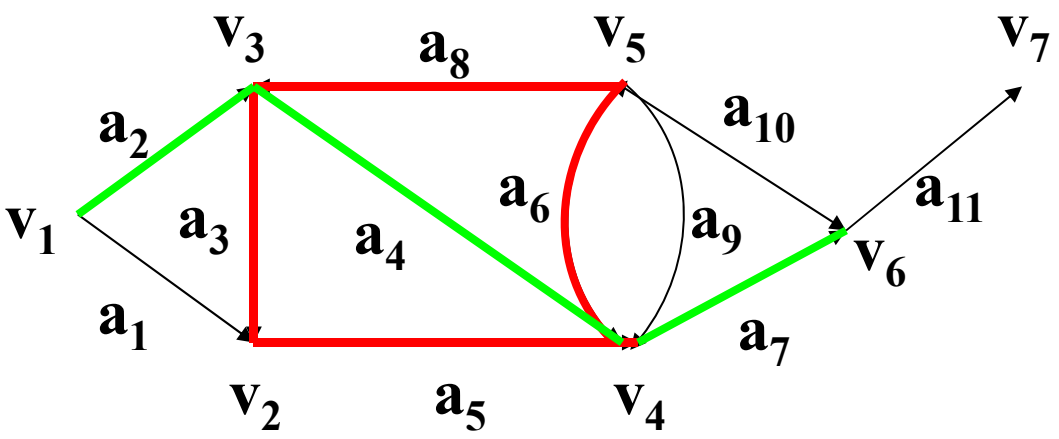
给定一个图G=(V,E)，如果图G'=(V',E')，使V'=V及E'⊆E，则称G'是G的一个支撑子图。



- 基础图

给定一个有向图D=(V,A)，从D中去掉所有弧上的箭头，所得到的无向图称为基础图。记之为G(D)。





- 路
- 初等路
- 回路

$(v_1, a_2, v_3, a_4, v_4, a_7, v_6)$是从$v_1$到$v_6$的路。也是一条初等路

在上图中,$(v_3, a_3, v_2, a_5, v_4, a_6, v_5, a_8, v_3)$是一个回路。

- 简单有向图
- 多重有向图

- 权与网络

赋权图是指每条边都有一个（或多个）实数对应的图，这个（些）实数称为这条边的权（每条边可以具有多个权）。赋权图在实际问题中非常有用。根据不同的实际情况，权数的含义可以各不相同。例如，可用权数代表两地之间的实际距离或行车时间，也可用权数代表某工序所需的加工时间等。

在实际应用中，给定一个图**G=（V，E）**或有向图**D=（V，A）**，在**V**中指定两个点，一个称为始点（或发点），记作v_1，一个称为终点（或收点），记作v_n，其余的点称为中间点。对每一条弧 $(v_i, v_j) \in A$，对应一个数 w_{ij} ，称为弧上的“权”。通常把这种赋权的图称为网络。

- 图的矩阵表示

对于网络（赋权图）G=（V，E），其中边 (v_i, v_j) 有权 w_{ij}，构造矩阵 $A=(a_{ij})_{n\times n}$，其中：

$$a_{ij}=\begin{cases} w_{ij} & (v_i, v_j)\in E \\ 0 & (v_i, v_j)\notin E \end{cases}$$

称矩阵A为网络G的权矩阵。

设图G=（V，E）中顶点的个数为n，构造一个矩阵 $A=(a_{ij})_{n\times n}$，其中：

$$a_{ij}=\begin{cases} 1 & (v_i, v_j)\in E \\ 0 & (v_i, v_j)\notin E \end{cases}$$

称矩阵A为网络G的邻接矩阵。

•图的矩阵表示

对于网络（赋权图）G=（V，E），其中边 (v_i, v_j) 有权 w_{ij}.

构造矩阵 $A=(a_{ij})_{n\times n}$，当G为赋权图时

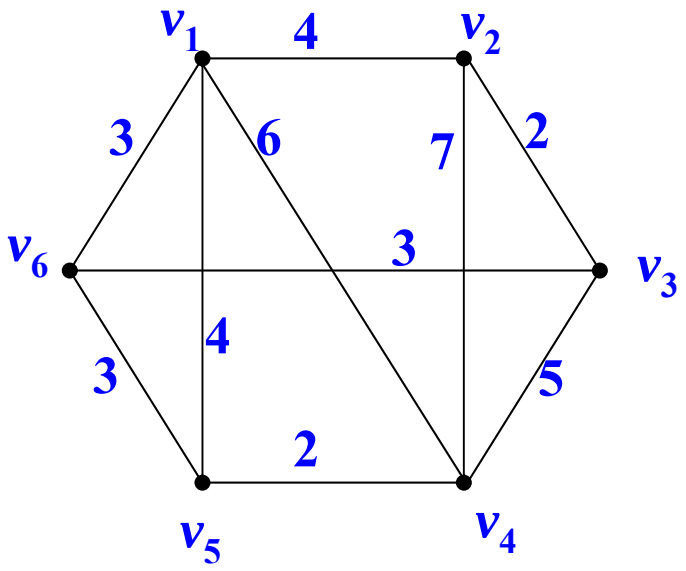
$$a_{ij}=\begin{cases} w_{ij}, \text{ 当}v_i\text{与}v_j\text{之间有边时，即}(v_i,v_j)\in E \\ 0\text{或}\infty, \text{ 当}v_i\text{与}v_j\text{之间无边时, 即}(v_i,v_j)\notin E. \end{cases}$$

称矩阵A为网络G的权矩阵。

设图G=（V，E）中顶点的个数为n，构造矩阵 $A=(a_{ij})_{n\times n}$，其中：

$$a_{ij}=\begin{cases} 1 & (v_i,v_j)\in E \\ 0 & (v_i,v_j)\notin E \end{cases}$$

称矩阵A为网络G的邻接矩阵。



图的矩阵表示

权矩阵为：

$$A=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}\begin{bmatrix} 0 & 4 & 0 & 6 & 4 & 3 \\ 4 & 0 & 2 & 7 & 0 & 0 \\ 0 & 2 & 0 & 5 & 0 & 3 \\ 6 & 7 & 5 & 0 & 2 & 0 \\ 4 & 0 & 0 & 2 & 0 & 3 \\ 3 & 0 & 3 & 0 & 3 & 0 \end{bmatrix}$$

$$\begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix}$$

邻接矩阵为：

$$B=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix}$$

采用邻接矩阵表示图，直观方便，通过查邻接矩阵元素的值可以很容易地查找图中任两个顶点 v_i 和 v_j 之间有无边，以及边上的权值。当图的边数 m 远小于顶点数 n 时，邻接矩阵表示法会造成很大的空间浪费。

- 稀疏矩阵表示法

稀疏矩阵是指矩阵中零元素很多，非零元素很少的矩阵。对于稀疏矩阵，只要存放非零元素的行标、列标、非零元素的值即可，可以按如下方式存储（非零元素的行地址，非零元素的列地址），非零元素的值。

在**Matlab**中无向图和有向图邻接矩阵的使用上有很大差异。

对于有向图，只要写出邻接矩阵，直接使用**Matlab**的命令 **sparse**命令，就可以把邻接矩阵转化为稀疏矩阵的表示方式。

对于无向图，由于邻接矩阵是对称阵，**Matlab**中只需使用邻接矩阵的下三角元素，即**Matlab**只存储邻接矩阵下三角元素中的非零元素。

稀疏矩阵只是一种存储格式。**Matlab**中，普通矩阵使用**sparse**命令变成稀疏矩阵，稀疏矩阵使用**full**命令变成普通矩阵。

1.2 基本定理

- 定理1

图G=(V,E)中，所有点的次之和是边数的两倍，即

$$\sum_{v \in V} d(v) = 2q$$

- 定理2

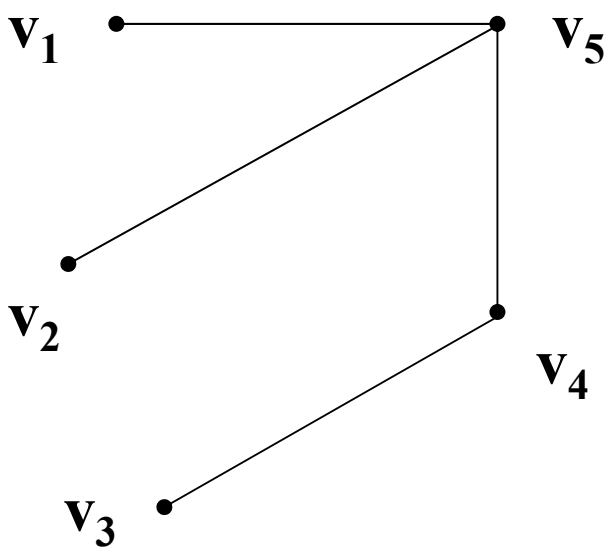
任一图中奇点的个数为偶数。

1.3 树的相关定义与定理

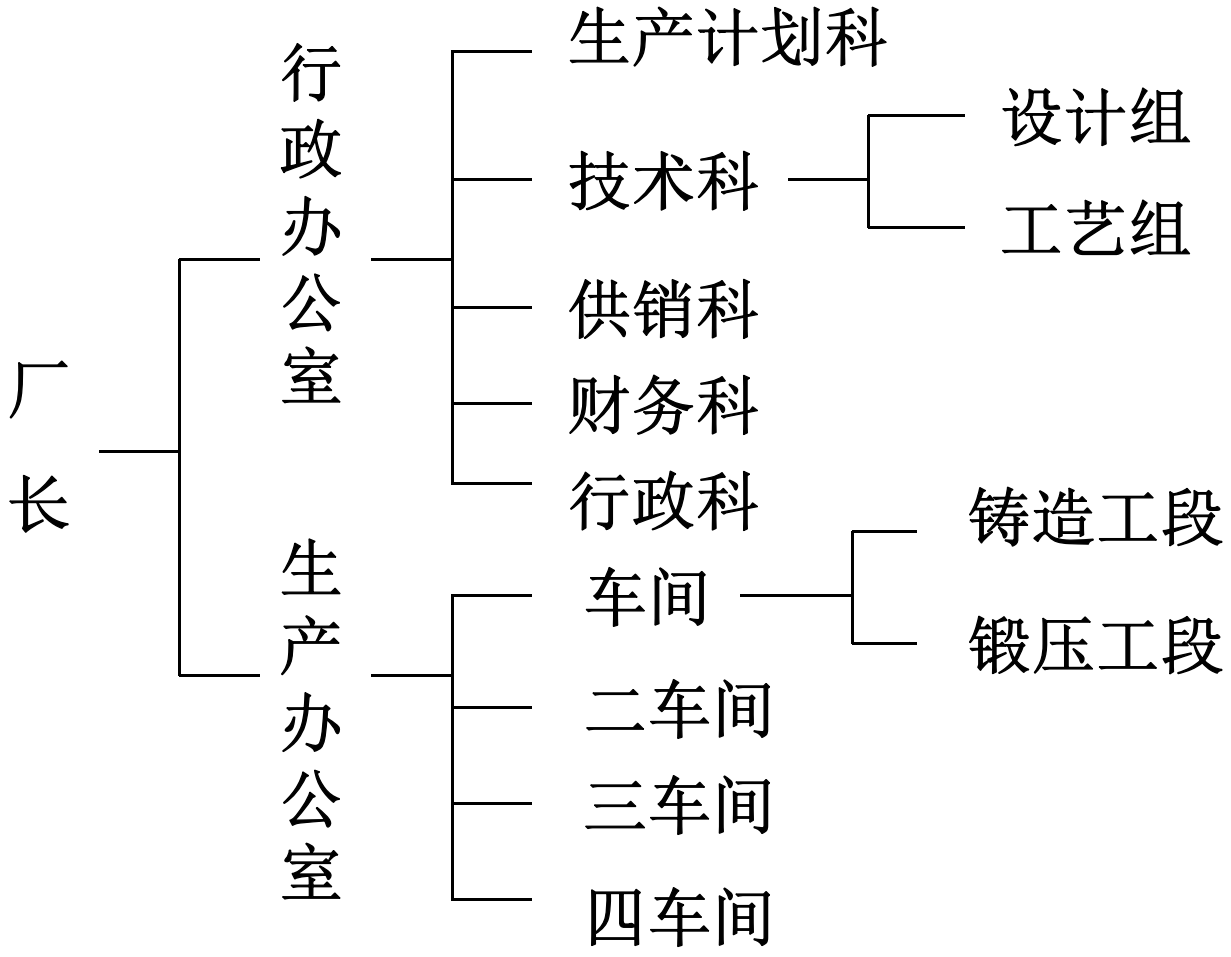
树的定义：一个无圈的连通图。

例1　在五个城市之间架设电话线，要求任两个城市之间都可以相互通话（允许通过其他城市），并且电话线的根数最少。

用v_1,v_2,v_3,v_4,v_5代表五个城市,如果在某两个城市之间架设电话线,则在相应的两点之间联一条边,这样一个电话线网就可以用一个图来表示。显然,这个图必须是连通的,而且是不含圈的连通图。如右图所示。



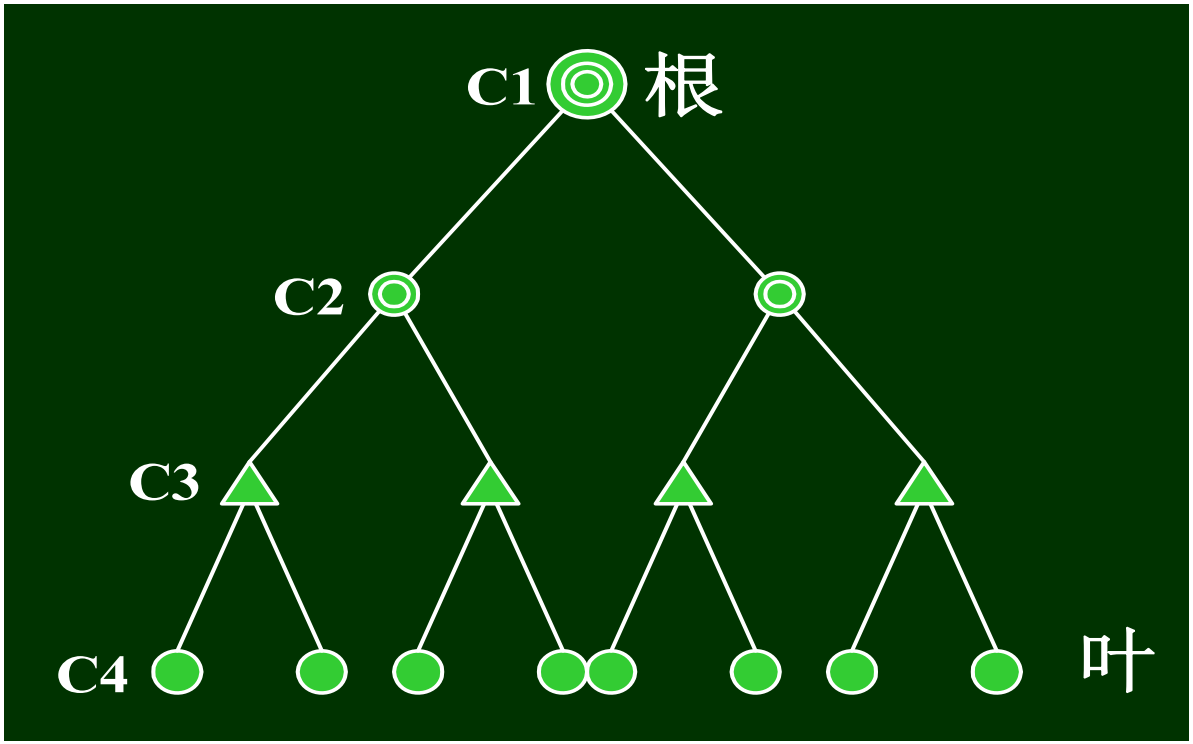
例2　某工厂的组织机构如下图所示



该厂的组织机构图就是一个树。

例3　树图

倒置的树，根(*root*)在上，叶(*leaf*)在下



二、性质

定理1

设图G=(V,E) 是一个树，$p(G)\geq 2$，则G中至少有两个悬挂点。

定理2

图G=(V,E) 是一个树的充分必要条件是G中不含圈，且恰有$p-1$条边。

定理3

图G=(V,E)是一个树的充分必要条件是G是连通图，并且$q(G)=p(G)-1$。

定理4

图G是树的充分必要条件是任意两个顶点之间恰有一条链。

推论：

- 从一个树中去掉一条边，则余下的图是不连通的。
- 在树中不相邻的两个点间添上一条边，则恰好得到一个圈。

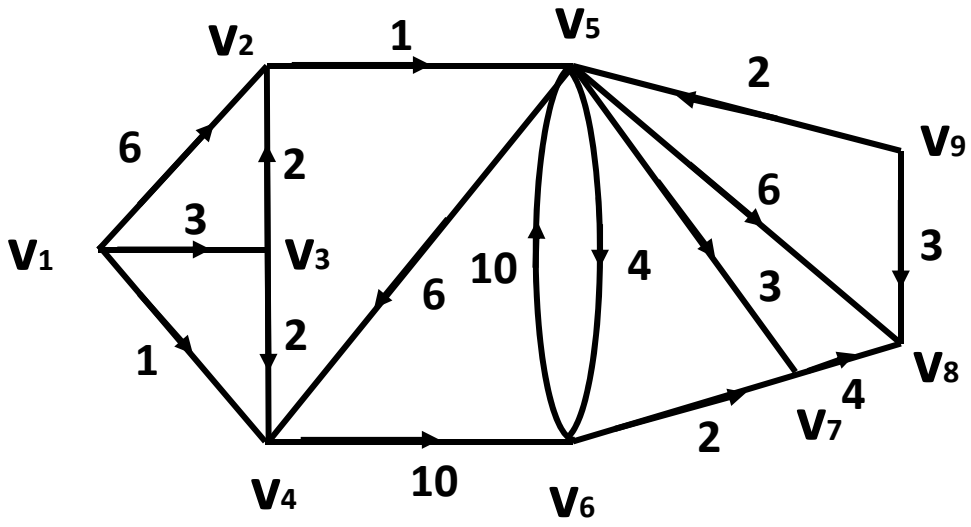
三、图的支撑树

定义：设图**T=(V,E')** 是图**G**的支撑子图，如果图**T=(V, E')** 是一个树，则称**T**是**G**的一个支撑树。

定理：图**G**有支撑树的充分必要条件是图**G**是连通的。

2　最短路问题

问题如下，给出了一个连接若干个城镇的单行铁路交通网，每弧旁的数字表示通过这条单行线的距离，现某人从v_1出发，通过这个网络到v_8，求使距离最小的旅行线路。



在图论中，最短路问题可归结为：

(1)给定一个赋权有向图D＝(V,A)及$W(a)=W_{ij}$；

(2)给定D中两个顶点v_s、v_t，P是D中从v_s到v_t的一条路；

(3)定义路P的权为P中所有弧的权之和，$W(P)=\Sigma W_{ij}$；

(4)求一条权最小的路P_0：

$$W(P_0)=\min W(P)$$

基本原理：最短路问题的特性

最短路线上的任一中间点到终（起）点的路线一定是该中间点到终（起）点的所有路线中最短的路线。

若求起点A到任一点G的最短路线，可先求A到G点的相邻前点的最短距离，在此基础上求出A到G点的最短距离。这样，最终求出起点到终点的最短距离。

常见的求解最短路问题的方法如下：

1. **Dijkstra**方法（迪科斯彻算法，荷兰计算机科学家**1959**提出）；
2. **Warshall-Floyd**算法（弗洛伊德算法，**1962**）；
3. 逐次逼近法.

实例：**CMCM94A**－公路选址问题。

最短路问题是重要的最优化问题之一，它不仅可以直接应用于解决生产实际的许多问题，如管道铺设、线路安排、厂区布局、设备更新等，而且经常被作为一个基本工具，用于解决其他优化问题。

- **Dijkstra算法**

基本原理：从起点v_s出发，按从近到远的顺序，逐步向外探寻最短路。在已知前点的最短距离基础上，求其后点的最短距离。

Dijkstra 最短路径算法

输入	图 $G=(V(G),E(G))$ 有一个源顶点 s 和一个汇顶点 t，以及对所有的边 $ij\in E(G)$ 的非负边长 c_{ij}.
输出	G 中从 s 到 t 的最短路径的长度.
第 0 步	从对每个顶点做临时标记 L 开始，做法如下：$L(s)=0$，且对除 s 外所有的顶点 $L(i)=\infty$.
第 1 步	找带有最小临时标记的顶点(如果有结，随机地取一个). 使该标记变成永久标记，意即该标记不再改变.
第 2 步	对每个没有永久标记但是又与带有永久标记的顶点相邻的顶点 j，按如下方法计算一个新的临时标记：$L(j)=\min\{L(i)+c_{ij}\}$，求最小是对所有带有永久标记的顶点 i 做的. 重复第 1 步和第 2 步，直到所有的顶点都打上了永久标记为止.

假设有向图有n个顶点，现需要求从顶点v_1到顶点v_n的最短路。设$W=(w_{ij})_{n\times n}$为邻接矩阵，其分量为

$$w_{ij}=\begin{cases}\text{边}v_iv_j\text{的权值}, & v_iv_j\in E,\\ \infty, & \text{其它},\end{cases}$$

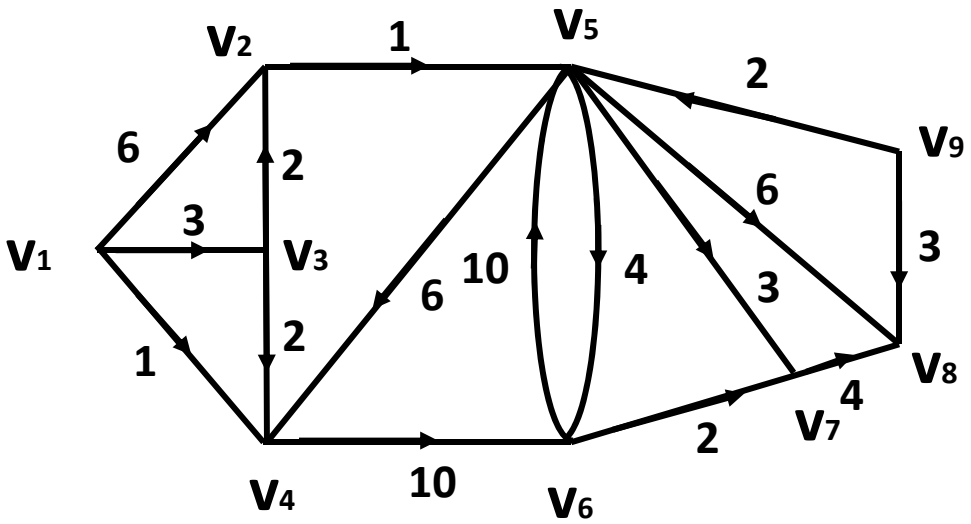
决策变量为x_{ij}，当$x_{ij}=1$，说明弧v_iv_j位于顶点v_1至顶点v_n的最短路上；否则$x_{ij}=0$。其数学规划表达式为

$$\min \sum_{v_iv_j\in E} w_{ij}x_{ij},$$

$$\text{s.t.} \sum_{\substack{j=1\\ v_iv_j\in E}}^{n} x_{ij}-\sum_{\substack{j=1\\ v_jv_i\in E}}^{n} x_{ji}=\begin{cases}1, & i=1,\\ -1, & i=n,\\ 0, & i\neq 1,n,\end{cases}$$

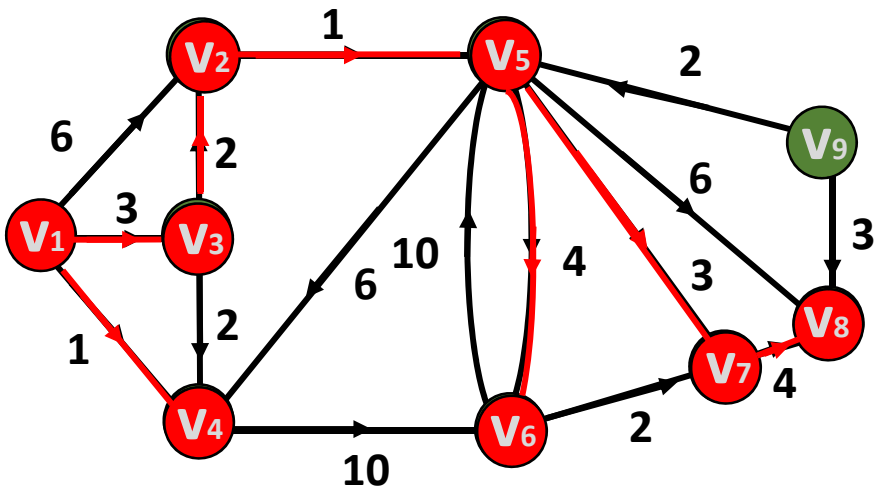
$$x_{ij}=0\text{或}1.$$

例1：

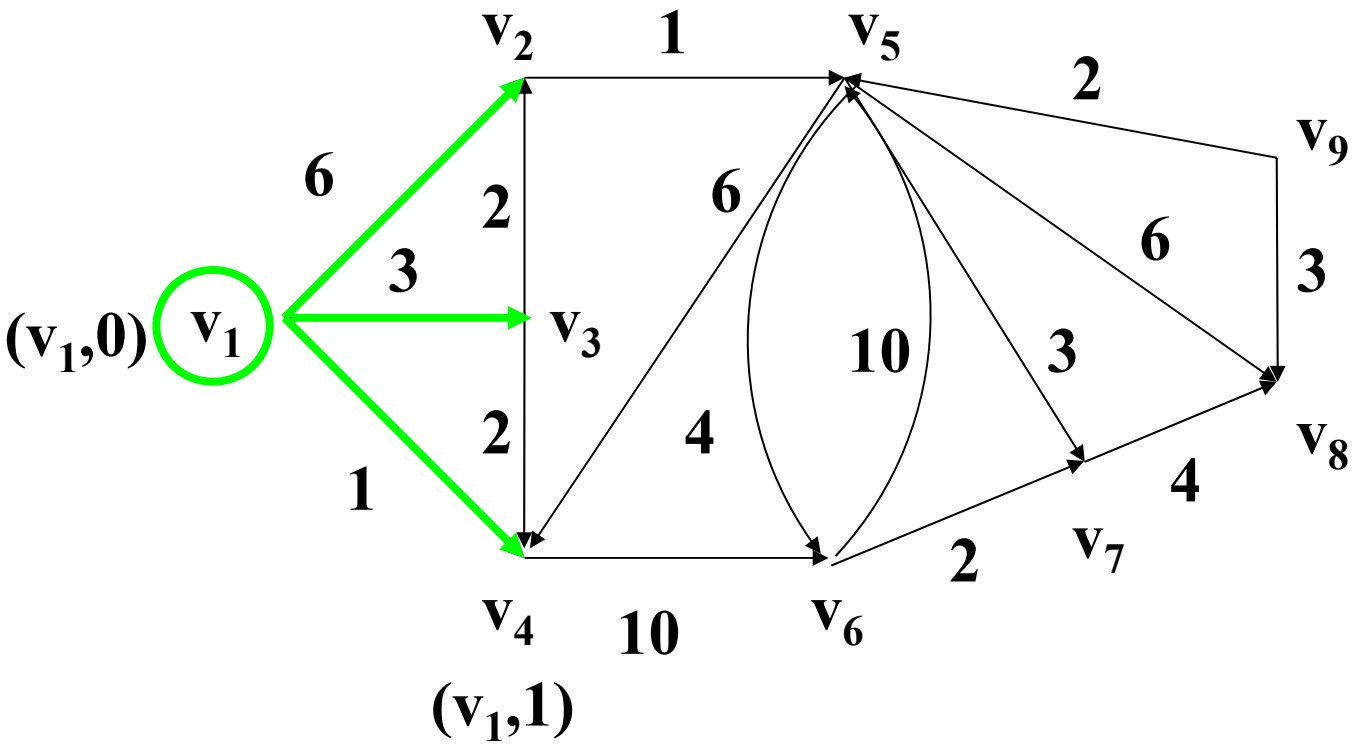


有向图

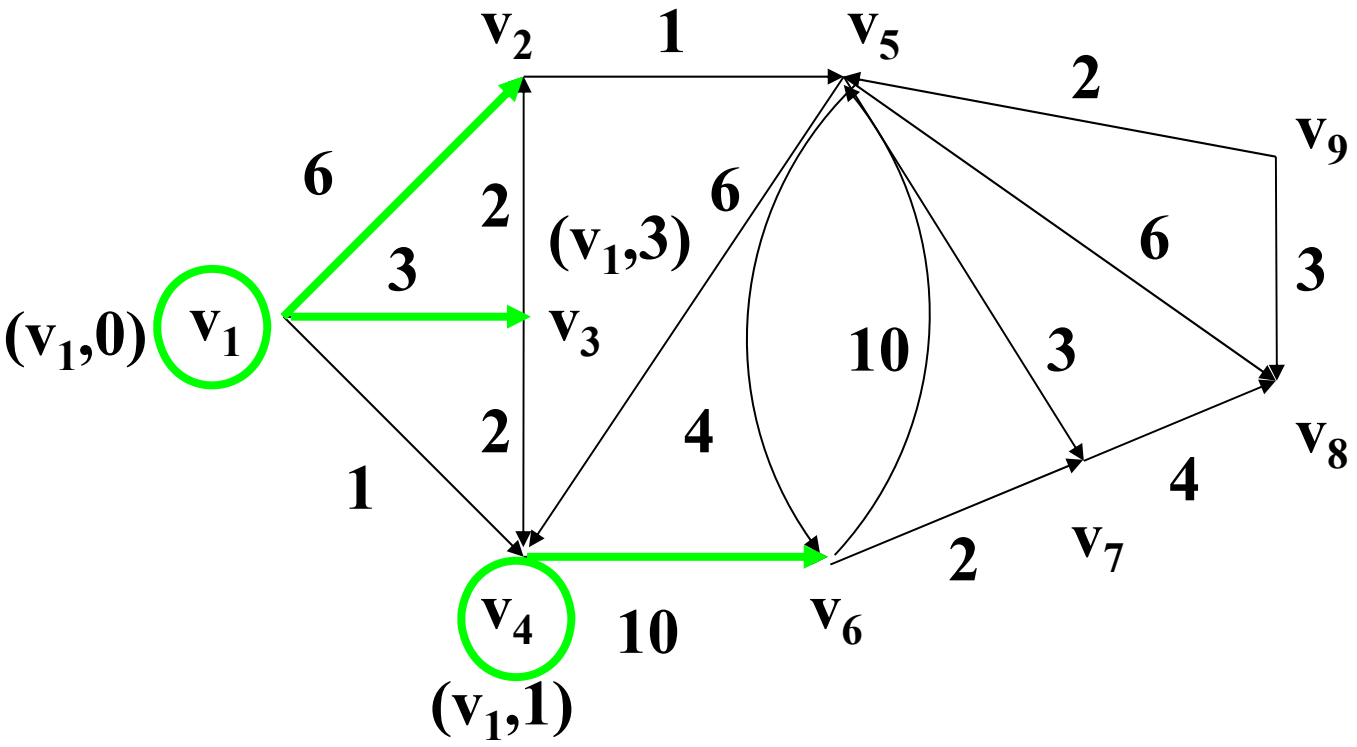
$$W=\begin{pmatrix}
0 & 6 & 3 & 1 & \infty & \infty & \infty & \infty & \infty \\
\infty & 0 & \infty & \infty & 1 & \infty & \infty & \infty & \infty \\
\infty & 2 & 0 & 2 & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & \infty & 0 & \infty & 10 & \infty & \infty & \infty \\
\infty & \infty & \infty & 6 & 0 & 4 & 3 & 6 & \infty \\
\infty & \infty & \infty & \infty & 10 & 0 & 2 & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & 0 & 4 & \infty \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\
\infty & \infty & \infty & \infty & 2 & \infty & \infty & 3 & 0
\end{pmatrix}$$



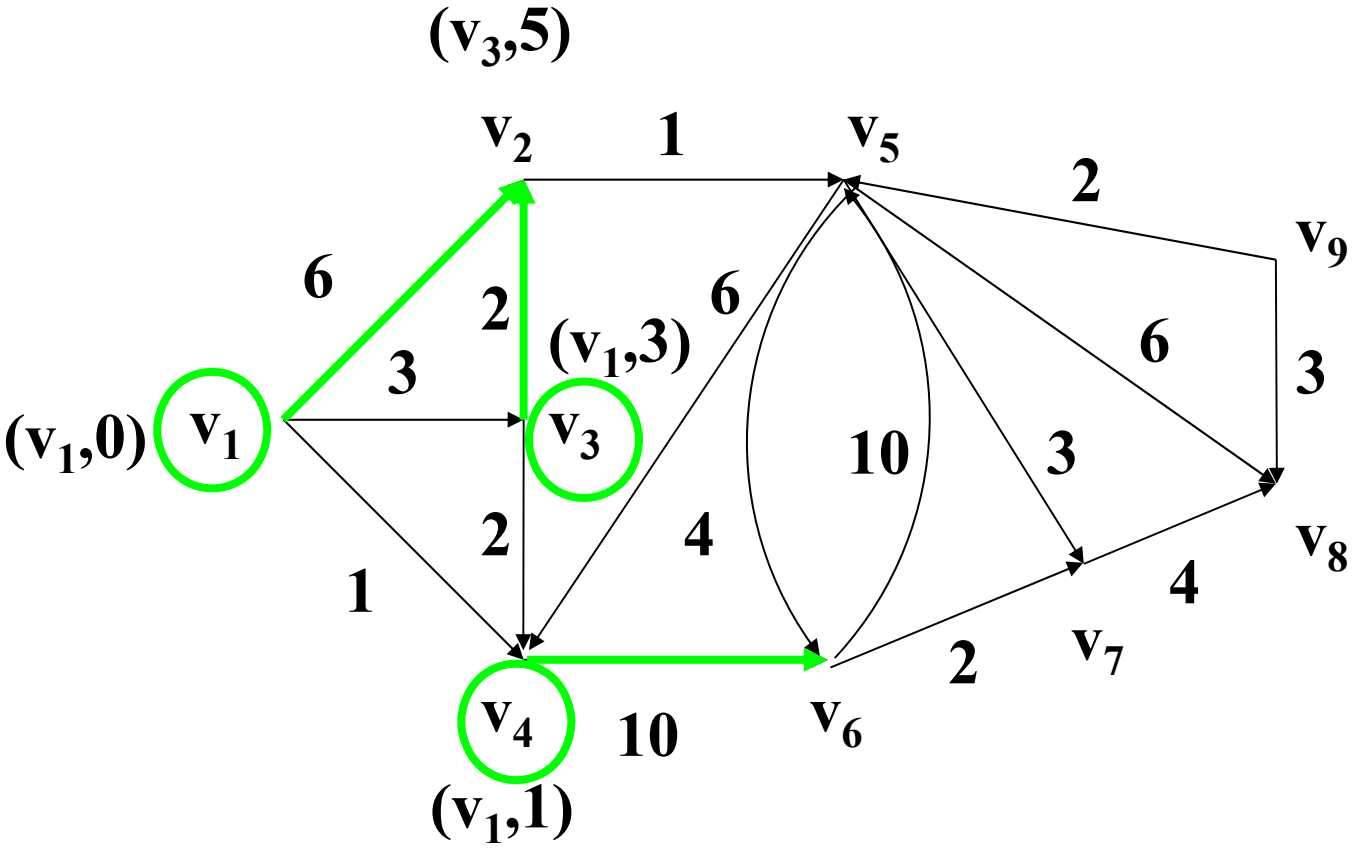
第一步：



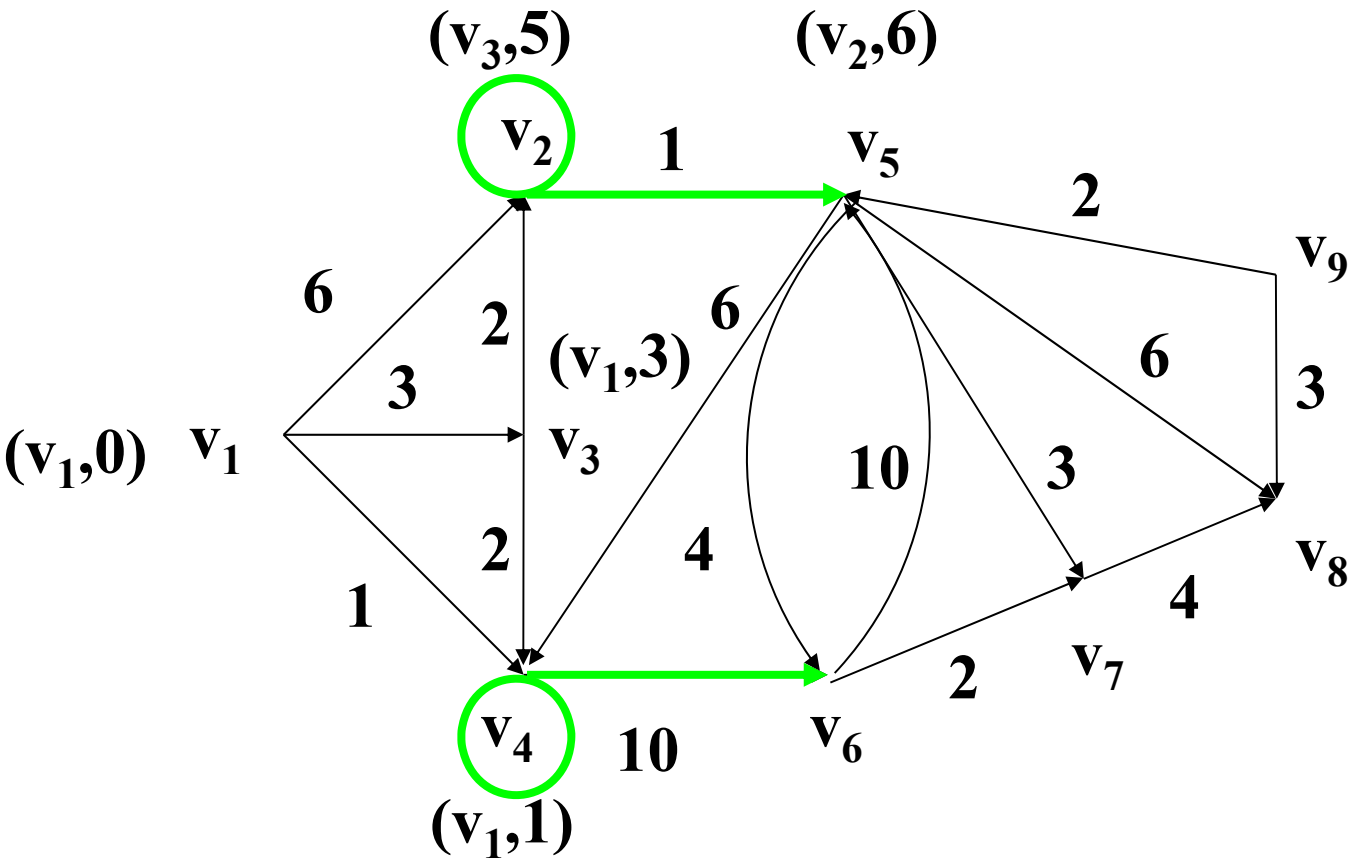
第二步：



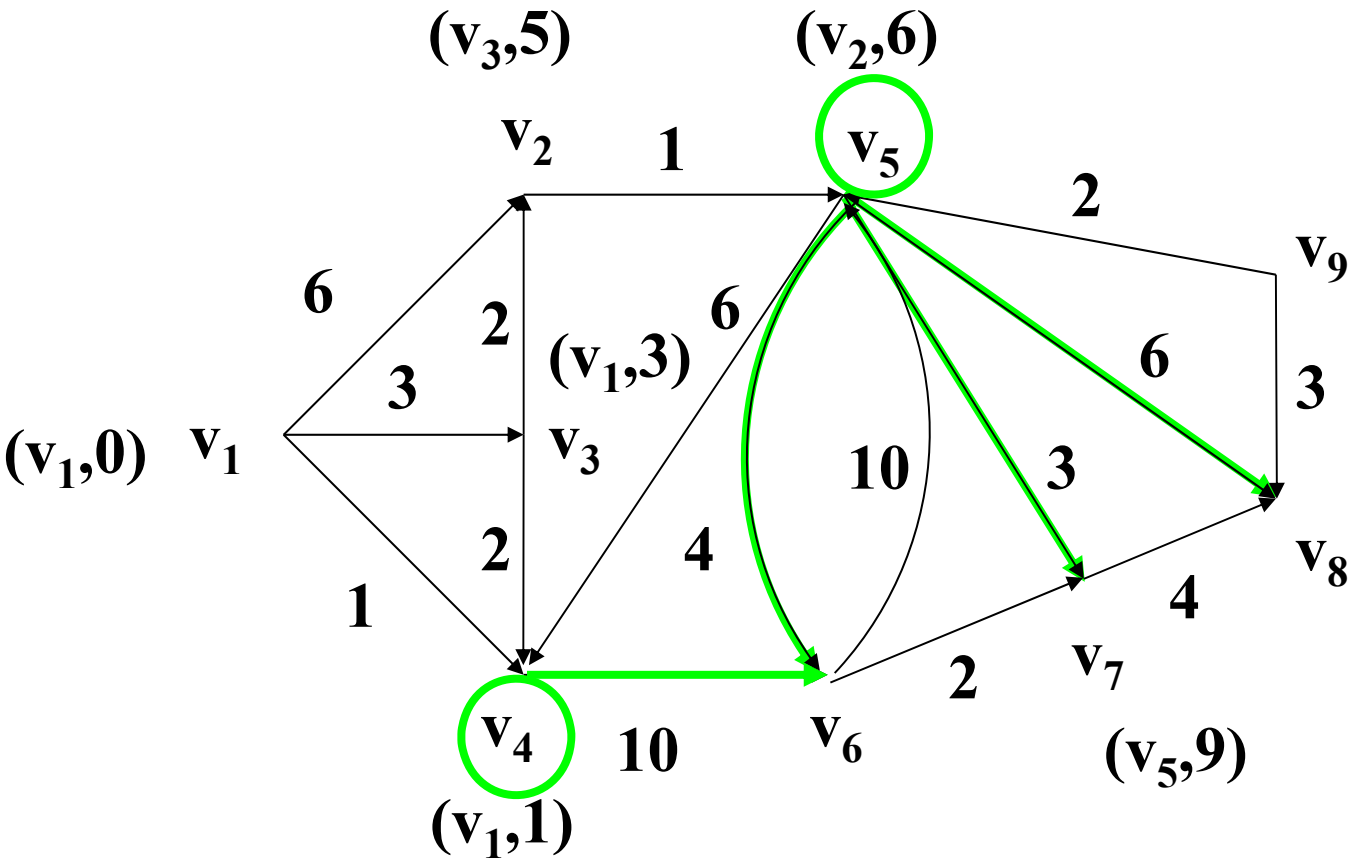
第三步：



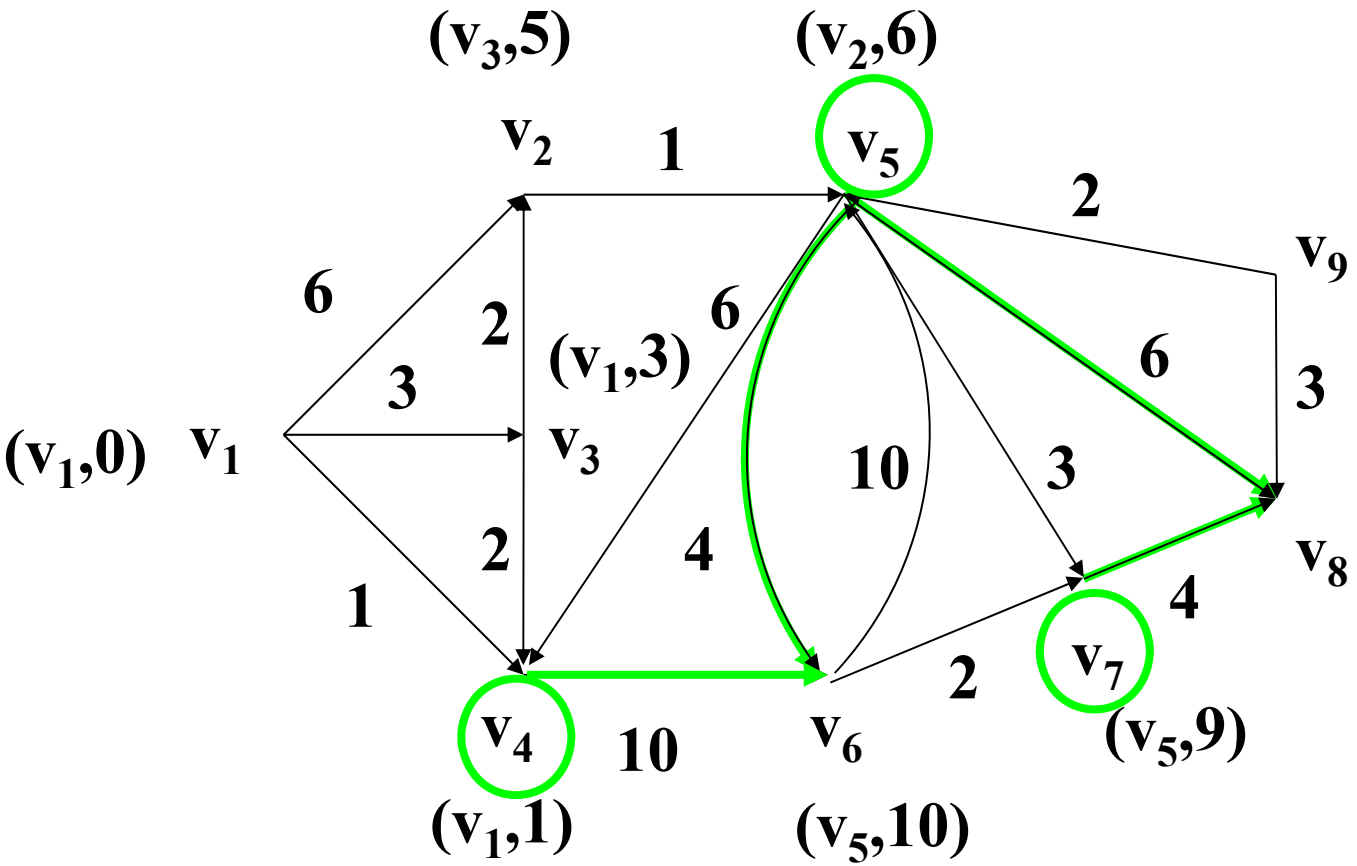
第四步：


(v3,5)
(v2,6)
v2
1
v5
2
v9
6
2
6
(v1,3)
6
3
3
(v1,0)
v1
v3
10
3
2
4
v8
1
4
2
v7
v4
10
v6
(v1,1)

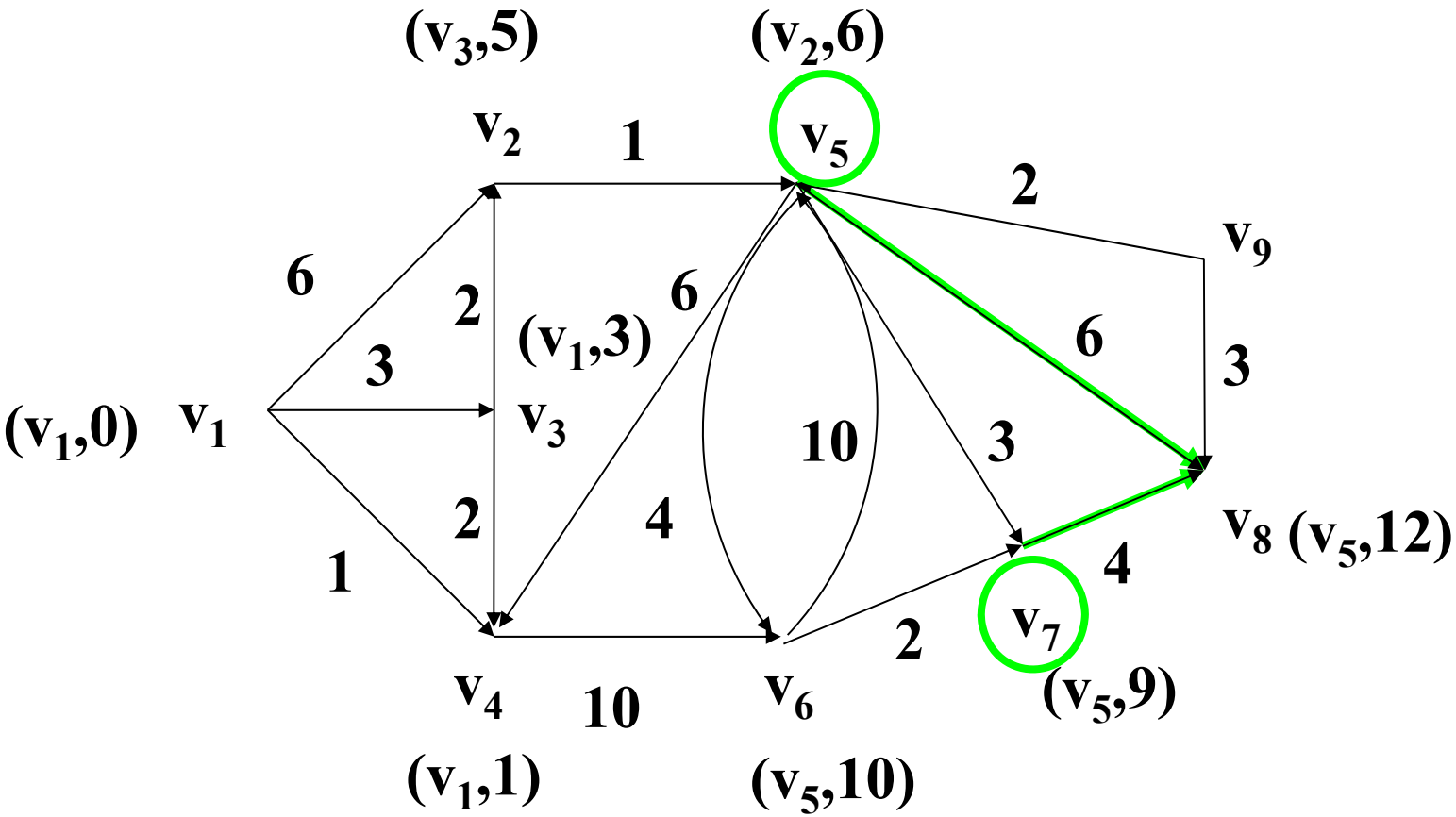
第五步：


(v3,5)
(v2,6)
v2
1
v5
2
v9
6
2
6
6
3
(v1,3)
3
(v1,0)
v1
v3
10
3
2
4
v8
1
4
2
v7
v4
10
v6
(v5,9)
(v1,1)

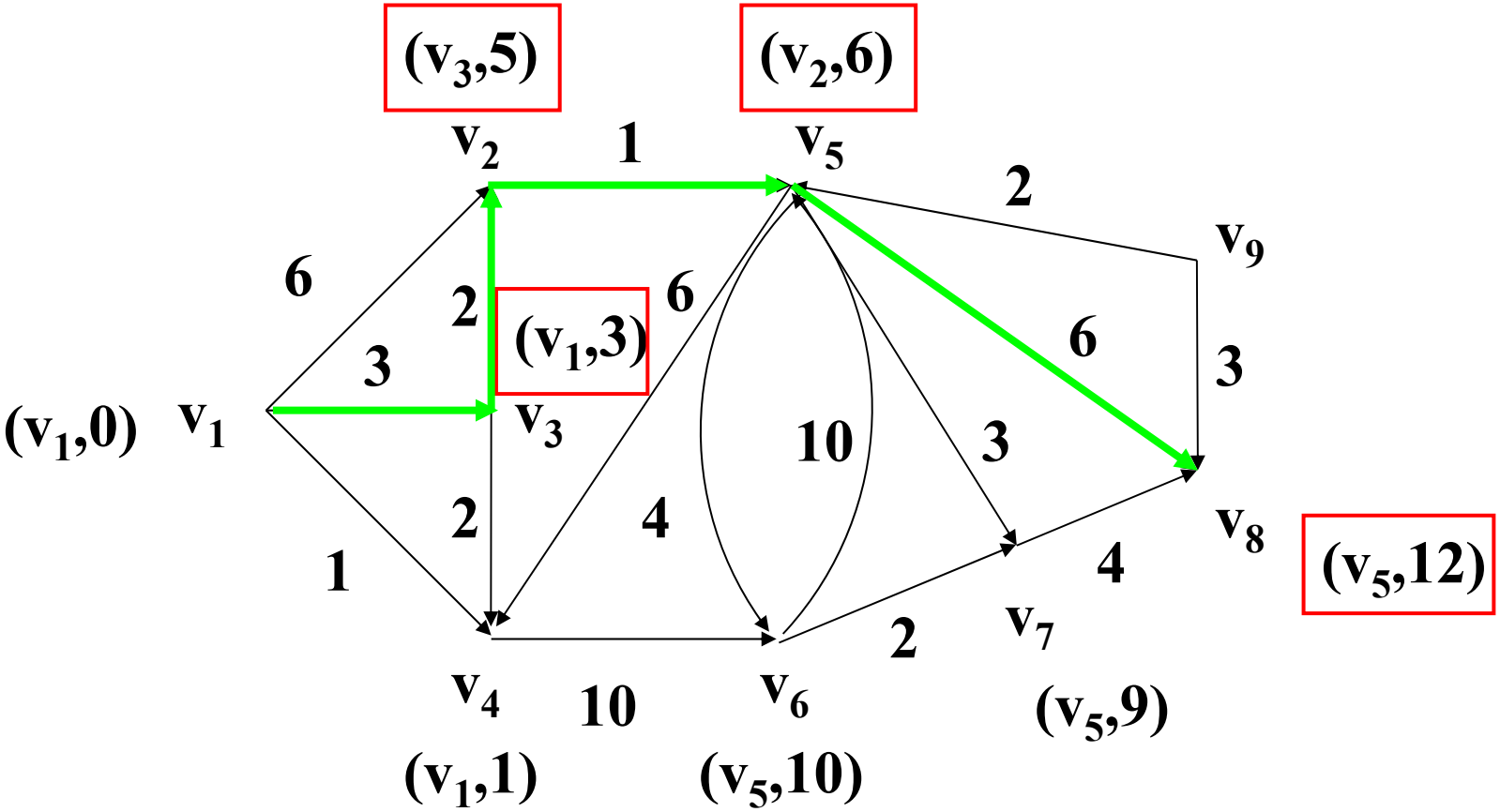
第六步：


(v3,5)
(v2,6)
v2
1
v5
2
v9
6
2
6
6
3
(v1,3)
3
(v1,0)
v1
v3
10
3
2
4
v8
1
4
v7
2
v4
10
v6
(v5,9)
(v1,1)
(v5,10)

第七步：


(v3,5)
(v2,6)
v2
1
v5
2
v9
6
2
6
6
3
(v1,3)
3
(v1,0)
v1
v3
10
3
2
4
v8 (v5,12)
1
4
v7
2
v4
10
v6
(v5,9)
(v1,1)
(v5,10)

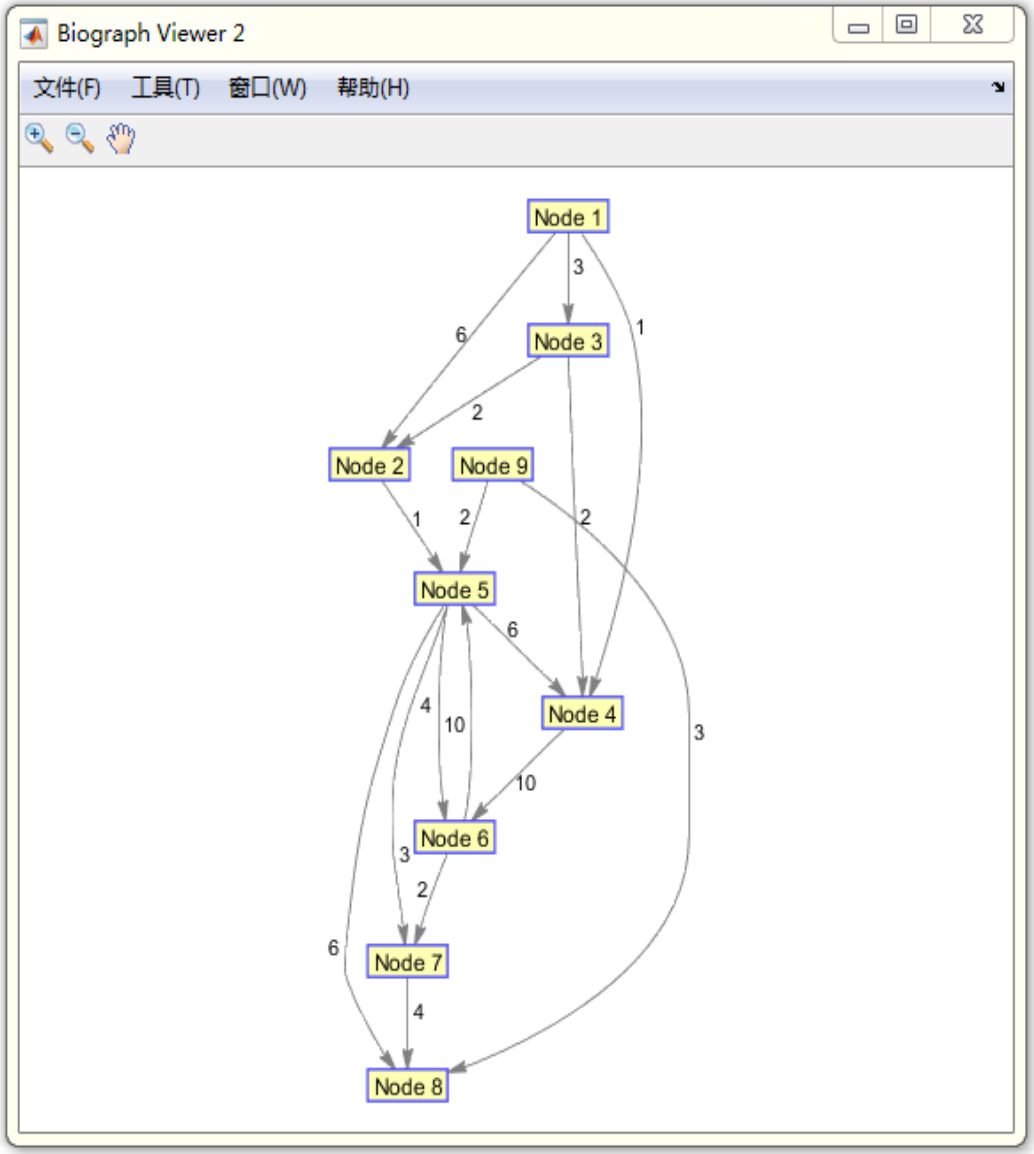
最后，找出v_1到v_8的最短路为：



缺点：不能解决存在负权图的最短路问题。

```
a(1,2)=6;a(1,3)=3;a(1,4
)=1;
a(2,5)=1;
a(3,2)=2;a(3,4)=2;
a(4,6)=10;
a(5,4)=6;a(5,6)=4;a(5,7
)=3;a(5,8)=6;
a(6,5)=10;a(6,7)=2;
a(7,8)=4;
a(9,5)=2;a(9,8)=3;
a(9,9)=0;
a=sparse(a);
```

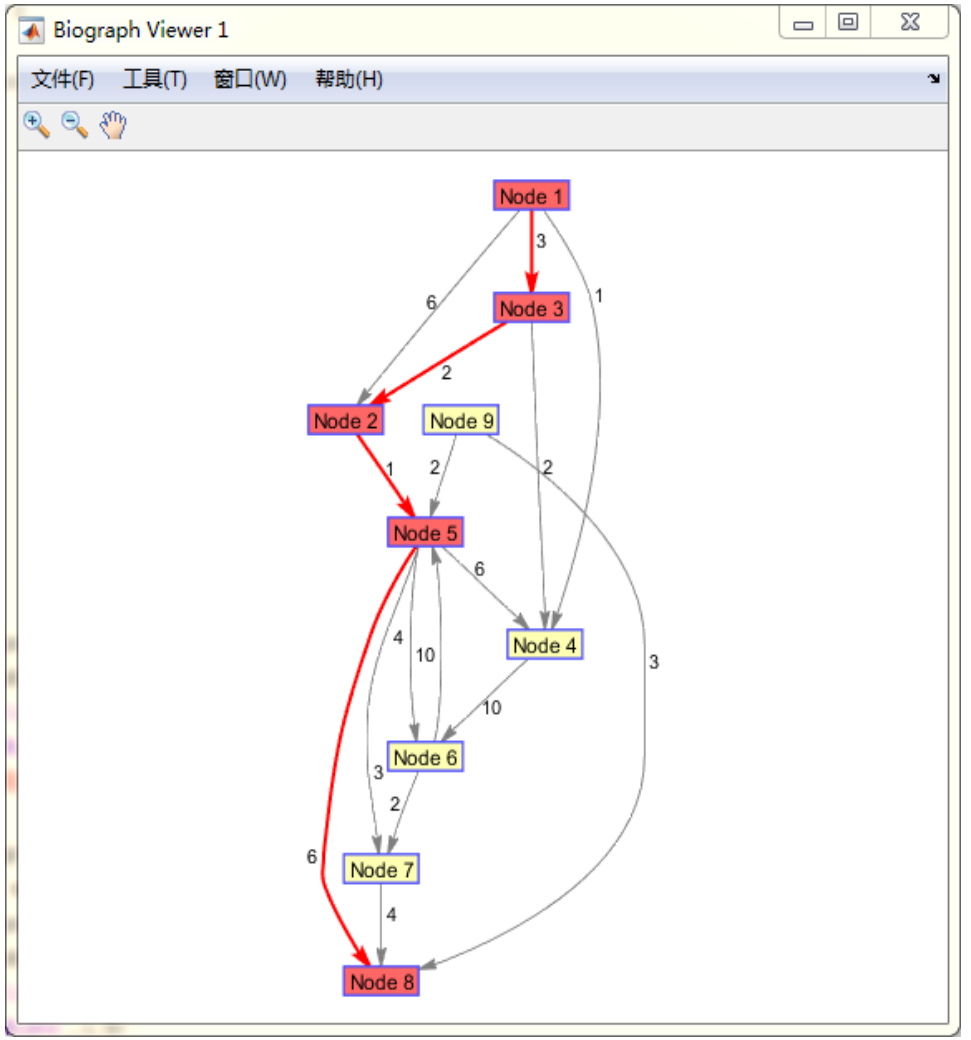
```
>> h = view(biograph(a,[],'ShowWeights','on'))
Biograph object with 9 nodes and 16 edges.
>>
```



```
[dist,path,pred] = graphshortestpath(a,1,8)
```

```
dist =
   12
path =
    1    3    2    5    8
pred =
    0    3    1    1    2    5    5    5   NaN
```

```
>> set(h.Nodes(path),'Color',[1 0.4 0.4])
edges =
getedgesbynodeid(h,get(h.Nodes(path),'ID'));
set(edges,'LineColor',[1 0 0])
set(edges,'LineWidth',1.5)
```



Matlab工具箱

```
[dist, path, pred] = graphshortestpath(G, S)
[dist, path, pred] = graphshortestpath(G, S, T)
```

其中：**G**为图所对应的赋权矩阵（表示为稀疏矩阵形式），**S**为起点，**T**为终点。

```
[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)
[...] = graphshortestpath(..., 'Method', MethodValue, ...)
```

- *DirectedValue*是用来表示图是有向图还是无向图，如果输入*false*，则接下来赋权矩阵*G*的上三角部分会被忽略。默认值为*true*，即有向图。
- *MethodValue*是用于选择求最短路的算法：‘Bellman-Ford’、‘BFS’、‘Acyclic’、‘Dijkstra’。默认算法为‘Dijkstra’。

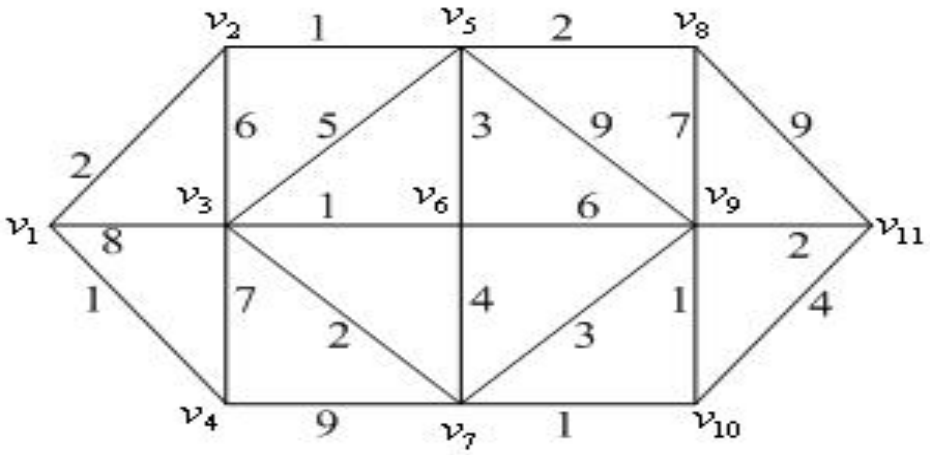
例2 某公司在六个城市$c_1, c_2, \cdots, c_6$中有分公司，从c_i到c_j的直接航程票价记在下述矩阵的(i, j)位置上。（∞表示无直接航路），请帮助该公司设计一张城市c_1到其他城市间的票价最便宜的路线图。

$$\begin{bmatrix} 0 & 50 & \infty & 40 & 25 & 10 \\ 50 & 0 & 15 & 20 & \infty & 25 \\ \infty & 15 & 0 & 10 & 20 & \infty \\ 40 & 20 & 10 & 0 & 10 & 25 \\ 25 & \infty & 20 & 10 & 0 & 55 \\ 10 & 25 & \infty & 25 & 55 & 0 \end{bmatrix}$$

```
clc;clear;
G(1,2)=50;G(1,4)=40;G(1,5)=25;G(1,6)=10;
G(2,3)=15;G(2,4)=20;G(2,6)=25;
G(3,4)=10;G(3,5)=20;
G(4,5)=10;G(4,6)=25;
G(5,6)=55;
G(6,6)=0;
G=sparse(G);
G=G';
[dist,path,pred] = graphshortestpath(G,1,'directed',false)
```

```
dist =
   0   35   45   35   25   10
path =
  [1]   [1x3 double]   [1x3 double]   [1x3 double]   [1x2 double]   [1x2 double]
pred =
   0    6    5    6    1    1
```

例 3（无向图的最短路问题）求图中v_1到v_{11}的最短路。



求得最短路径为

$$v_1 \to v_2 \to v_5 \to v_6 \to v_3 \to v_7 \to v_{10} \to v_9 \to v_{11}$$，

最短路径的长度为 13。

```
clc;clear;
a(1,2)=2;a(1,3)=8;a(1,4)=1;
a(2,3)=6;a(2,5)=1;
a(3,4)=7;a(3,5)=5;a(3,6)=1;a(3,7)=2;
a(4,7)=9;
a(5,6)=3;a(5,8)=2;a(5,9)=9;
a(6,7)=4;a(6,9)=6;
a(7,9)=3;a(7,10)=1;
a(8,9)=7;a(8,11)=9;
a(9,10)=1;a(9,11)=2;
a(10,11)=4;;
a=a';
[i,j,v]=find(a);
b=sparse(i,j,v,11,11);
[x,y,z]=graphshortestpath(b,1,11,'directed',false)
```

```
x =
  13
y =
   1   2   5   6   3   7  10   9  11
z =
   0   1   6   1   2   5   3   5  10   7   9
```

- Floyd算法

计算赋权图中各对顶点之间最短路径，显然可以调用Dijkstra 算法。具体方法是:每次以不同的顶点作为起点，用 Dijkstra 算法求出从该起点到其余顶点的最短路径，反复执行$n-1$次这样的操作，就可得到从每一个顶点到其它顶点的最短路径。这种算法的时间复杂度为$O(n^3)$。第二种解决这一问题的方法是由 Floyd, R. W.提出的算法，称之为 Floyd 算法。

对于赋权图$G=(V,E,A_0)$，其中顶点集$V=\{v_1,\cdots,v_n\}$

邻接矩阵

$$A_0=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},$$

这里

$$a_{ij}=\begin{cases} \text{权值，当}v_i\text{与}v_j\text{之间有边时，} \\ \infty\text{，当}v_i\text{与}v_j\text{之间无边时，} \end{cases} \qquad (i\neq j)$$

$$a_{ii}=0,\ i=1,2,\cdots,n.$$

对于无向图，A_0是对称矩阵，$a_{ij}=a_{ji}$。

Floyd 算法的基本思想是递推产生一个矩阵序列 $A_1,\cdots,A_k,\cdots,A_n$，其中矩阵 A_k 的第 i 行第 j 列元素 $A_k(i,j)$ 表示从顶点 v_i 到顶点 v_j 的路径上所经过的顶点序号不大于 k 的最短路径长度。

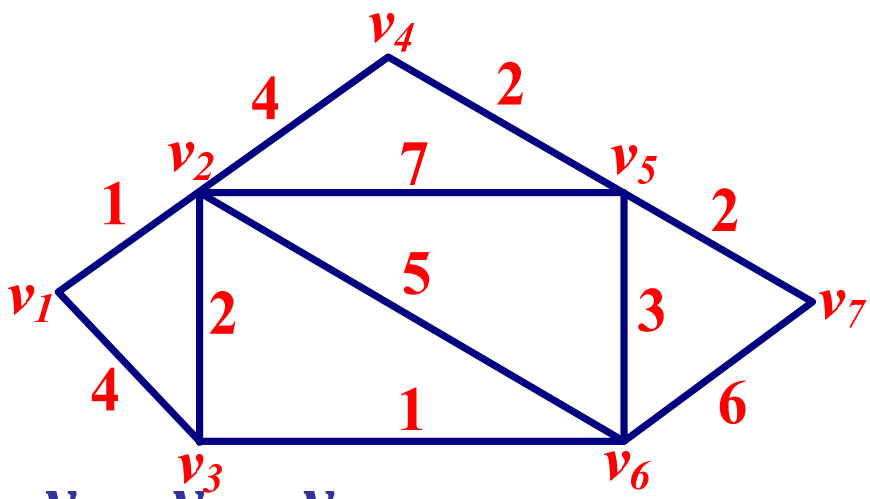
计算时用迭代公式

$$A_k(i,j)=\min(A_{k-1}(i,j),A_{k-1}(i,k)+A_{k-1}(k,j)),$$

k 是迭代次数，$i,j,k=1,2,\cdots,n$。

最后，当 $k=n$ 时，A_n 即是各顶点之间的最短通路值。

例4 用矩阵算法求图中v_1到v_7的最短路。



解 先将图表示成矩阵形式，将第一行起点标号0，并划去第一列。

		v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	v_1	0	1	4	∞	∞	∞	∞
	v_2	1	0	2	4	7	5	∞
	v_3	4	2	0	∞	∞	1	∞
	v_4	∞	4	∞	0	2	∞	∞
	v_5	∞	7	∞	2	0	3	2
	v_6	∞	5	1	∞	3	0	6
	v_7	∞	∞	∞	∞	2	6	0

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0 v_1	0	1	4	∞	∞	∞	∞
1 v_2	1	0	2+1	4+1	7+1	5+1	∞
v_3	4	2	0	∞	∞	1	∞
v_4	∞	4	∞	0	2	∞	∞
v_5	∞	7	∞	2	0	3	2
v_6	∞	5	1	∞	3	0	6
v_7	∞	∞	∞	∞	2	6	0

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\underline{0}$ v_1	0	(1)	4	∞	∞	∞	∞
$\underline{1}$ v_2	1	0	(3)	5	8	6	∞
$\underline{3}$ v_3	4	2	0	∞	∞	1 + 3	∞
v_4	∞	4	∞	0	2	∞	∞
v_5	∞	7	∞	2	0	3	2
v_6	∞	5	1	∞	3	0	6
v_7	∞	∞	∞	∞	2	6	0

		v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	v_1	0	1	4	∞	∞	∞	∞
1	v_2	1	0	3	5	8	6	∞
3	v_3	4	2	0	∞	∞	4	∞
	v_4	∞	4	∞	0	2	∞	∞
	v_5	∞	7	∞	2	0	3	2
4	v_6	∞	5	1	∞	3 + 4	0	6 + 4
	v_7	∞	∞	∞	∞	2	6	0

		v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	v_1	0	(1)	4	∞	∞	∞	∞
1	v_2	1	0	(3)	(5)	8	6	∞
3	v_3	4	2	0	∞	∞	(4)	∞
5	v_4	∞	4	∞	0	2 + 5	∞	∞
	v_5	∞	7	∞	2	0	3	2
4	v_6	∞	5	1	∞	7	0	10
	v_7	∞	∞	∞	∞	2	6	0

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0 v_1	0	1	4	∞	∞	∞	∞
1 v_2	1	0	3	5	8	6	∞
3 v_3	4	2	0	∞	∞	4	∞
5 v_4	∞	4	∞	0	7	∞	∞
7 v_5	∞	7	∞	2	0	3	2 + 7
4 v_6	∞	5	1	∞	7	0	10
v_7	∞	∞	∞	∞	2	6	0

- 下面再找出从v_1到v_7的最短通路。用逆推的方法，在终点v_7所在的列中，圈起来的9在第五行，即知v_7前面的点是v_5，在找v_5列，圈起来的7有两个，先随取一个，如第4行，则前面的点是v_4，则找v_4列，圈起来的是第二行，则v_4前的点是v_2，再找v_2列，圈起来的点在第一行，则前面的点是v_1，即起点，个可以找到v_1到v_7的一条最短通路是：
- $$v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7。$$

Matlab工具箱

求图中所有顶点对之间的最短距离：

[*dist*] = graphallshortestpaths(*G*)
[*dist*] = graphallshortestpaths(*G*, ...'Directed', *DirectedValue*, ...)

例2：

```
clc;clear;
G(1,2)=50;G(1,4)=40;G(1,5)=25
;G(1,6)=10;
G(2,3)=15;G(2,4)=20;G(2,6)=25;
G(3,4)=10;G(3,5)=20;
G(4,5)=10;G(4,6)=25;
G(5,6)=55;
G(6,6)=0;
G=sparse(G);
G=G'+G;
dist=graphallshortestpaths(G)
```

```
dist =
     0    35    45    35    25    10
    35     0    15    20    30    25
    45    15     0    10    20    35
    35    20    10     0    10    25
    25    30    20    10     0    35
    10    25    35    25    35     0
```

- 最短路问题 – 两点说明

最长路问题可以转化为最短路问题，把弧上的费用反号即可.

➢必须指出：目前为止，一切最短路算法都只对不含负有向圈的网络有效. 对于含负有向圈的网络，最短路问题是NP困难的.

➢因此，这里除非特别说明，一律假定网络不包含负有向圈.

无向网络上的最短路问题一般可以转化为有向网络上的问题.

➢如果所有弧上的权全为非负（或非正）数，只需将无向图的一条边代之以两条对称的有向弧即可.

➢如果弧上的权有负有正，一般来说问题要复杂得多。

3　最小生成树问题

3.1　基本概念

连通的无圈图叫做树，记之为T；其度为 1 的顶点称为叶子顶点；显然有边的树至少有两个叶子顶点。

若图$G=(V(G),E(G))$和树$T=(V(T),E(T))$满足$V(G)=V(T)$，$E(T)\subset E(G)$，则称T是G的生成树。图G连通的充分必要条件为G有生成树，一个连通图的生成树的个数很多。

树有下面常用的五个充要条件。

定理 （1）$G=(V,E)$是树当且仅当G中任二顶点之间有且仅有一条轨道。

（2）G是树当且仅当G无圈，且$|E|=|V|-1$。

（3）G是树当且仅当G连通，且$|E|=|V|-1$。

（4）G是树当且仅当G连通，且$\forall e\in E$，$G-e$不连通。

（5）G是树当且仅当G无圈，$\forall e\notin E$，$G+e$恰有一个圈。

3.2 最小生成树

欲修筑连接n个城市的铁路，已知i城与j城之间的铁路造价为c_{ij}，设计一个线路图，使总造价最低。

上述问题的数学模型是在连通赋权图上求权最小的生成树。赋权图的具最小权的生成树叫做最小生成树。

背景：筑路选线问题　　欲修筑连接n个城市的铁路，已知i城与j城之间的铁路造价为C_{ij}。设计一个线路图，使总造价最低。

分析：选线问题的数学模型是在连通加权图上求权最小的连通生成子图。显然，权最小的连通生成子图是一个生成树，即求取连通加权图上的权最小的生成树，这就归结为最小生成树问题。这个问题可由克罗斯克尔(Kruskal)算法、普莱姆(Prim)算法或者破圈法解决。

思路：从“边”着手选最小生成树（kruskal，破圈），从“点”着手（prim）。

3.2.1 prim 算法构造最小生成树

构造连通赋权图$G=(V,E,W)$的最小生成树，设置两个集合P和Q,其中P用于存放G的最小生成树中的顶点，集合Q存放G的最小生成树中的边。令集合P的初值为$P=\{v_1\}$（假设构造最小生成树时，从顶点v_1出发），集合Q的初值为$Q=\Phi$（空集）。prim 算法的思想是，从所有$p\in P$ $v\in V-P$的边中，选取具有最小权值的边pv，将顶点v加入集合P中，将边pv加入集合Q中，如此不断重复，直到$P=V$时，最小生成树构造完毕，这时集合Q中包含了最小生成树的所有边。

prim 算法如下

（1） $P=\{v_1\}$，$Q=\Phi$；

（2）while $P \sim = V$

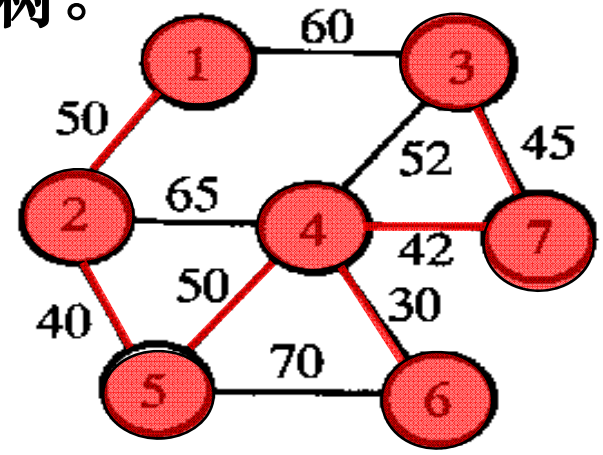
找最小边 pv，其中 $p \in P, v \in V-P$；

$P=P+\{v\}$；

$Q=Q+\{pv\}$；

end

例 5 用 prim 算法求右图的最小生成树。



求得最小生成树的边集为

$\{v_1v_2, v_2v_5, v_5v_4, v_4v_6, v_4v_7, v_7v_3\}$。

3.2.2 Kruskal 算法构造最小生成树

科茹斯克尔（Kruskal）算法是一个好算法。Kruskal算法如下

（1）选$e_1 \in E(G)$，使得e_1是权值最小的边。

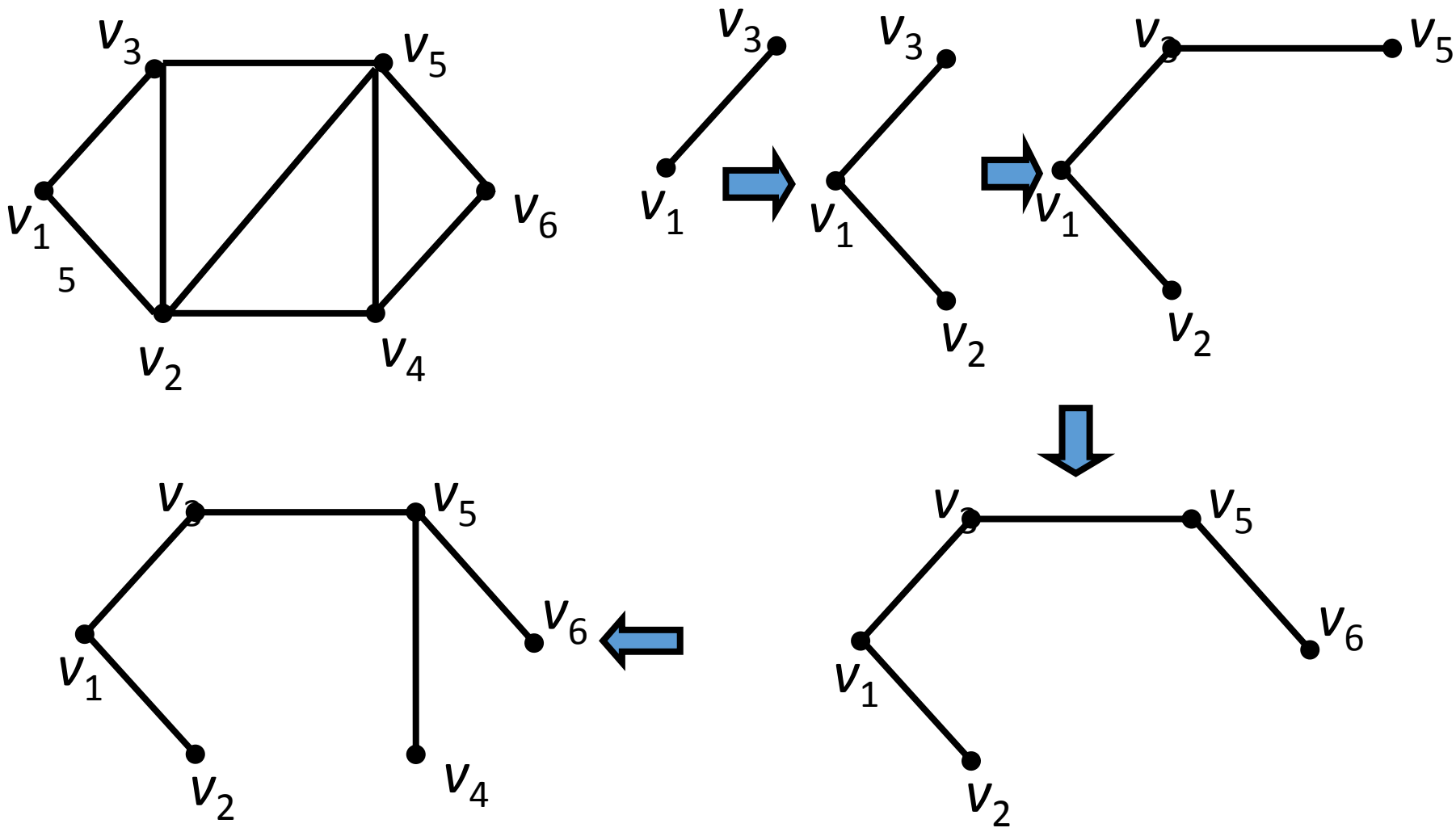
（2）若$e_1, e_2, \cdots, e_i$已选好，则从$E(G)-\{e_1, e_2, \cdots, e_i\}$中选取$e_{i+1}$，使得

i）$\{e_1, e_2, \cdots, e_i, e_{i+1}\}$中无圈，且

ii）e_{i+1}是$E(G)-\{e_1, e_2, \cdots, e_i\}$中权值最小的边。

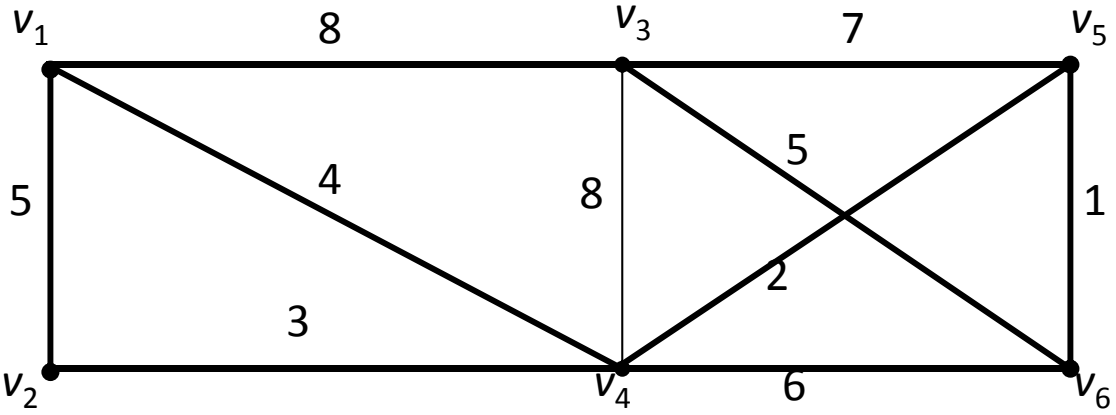
（3）直到选得$e_{|V|-1}$为止。

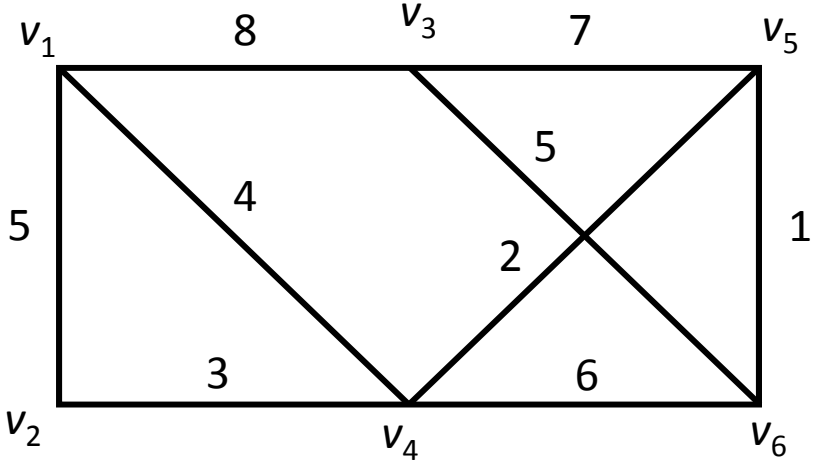
避圈法（Kruskal）

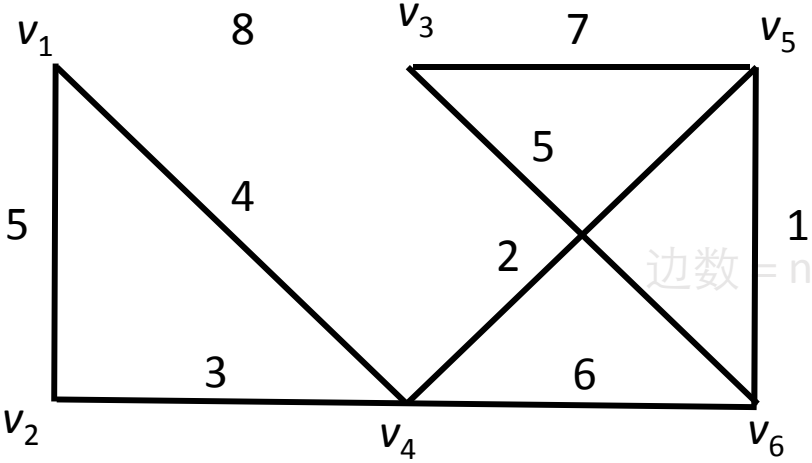


3.2.3破圈法（1975年管梅谷）

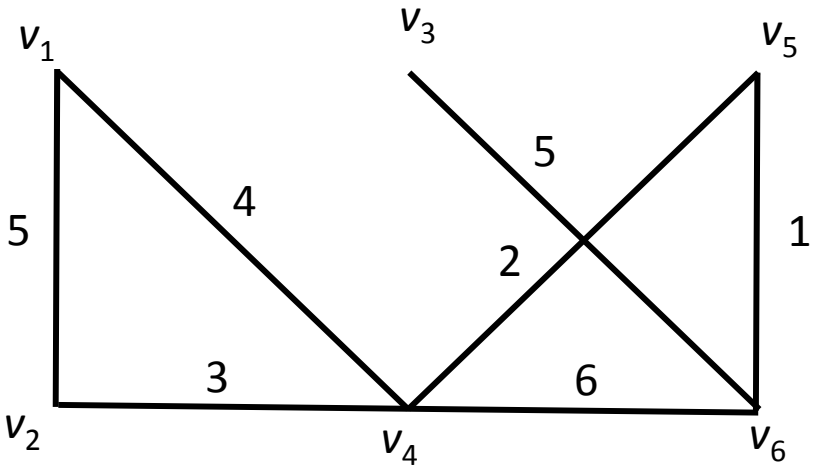
任取一圈，去掉圈中最长边，直到无圈。

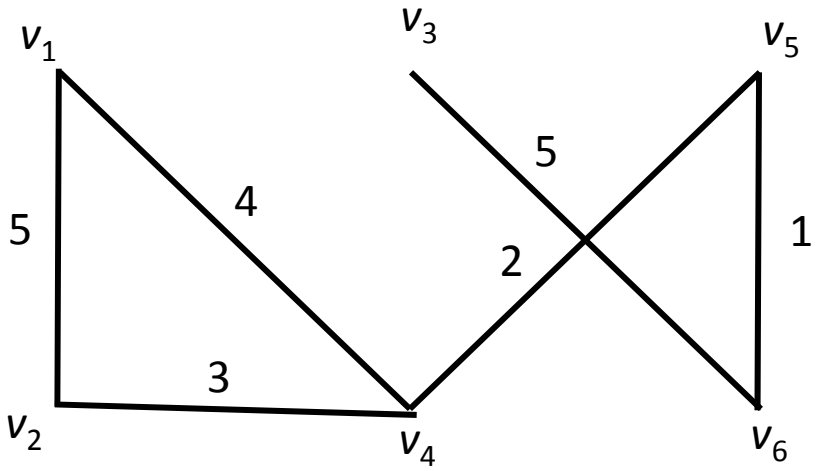


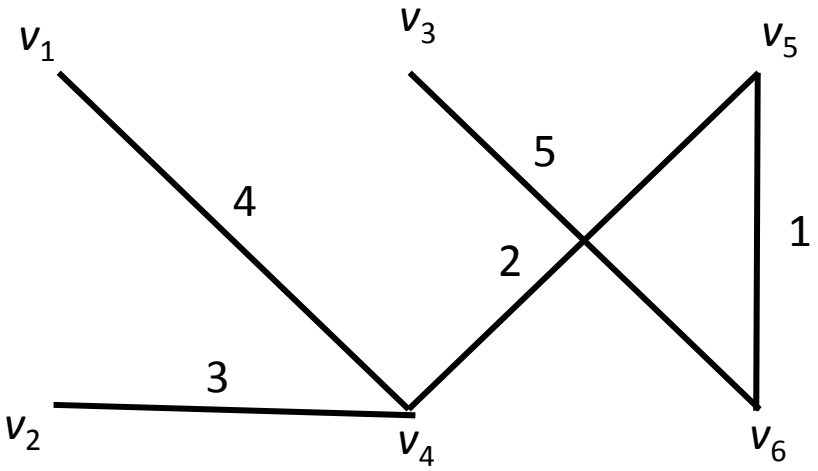




边数 = n-1=5







得到最小树：

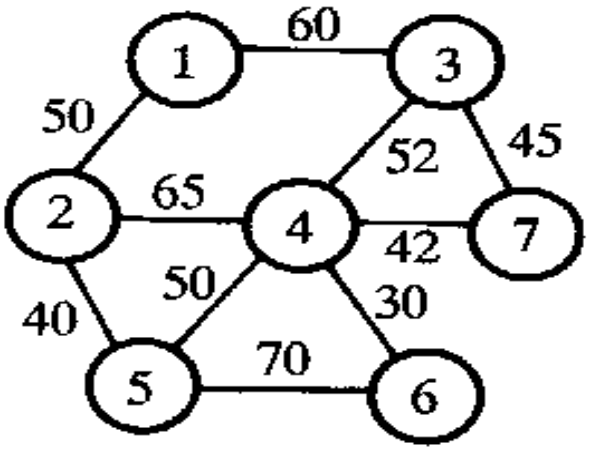
Min C(T)=15

Matlab工具箱

[*Tree*, *pred*] = graphminspantree(*G*)

[*Tree*, *pred*] = graphminspantree(*G*, *R*)

[*Tree*, *pred*] = graphminspantree(..., 'Method', *MethodValue*, ...)



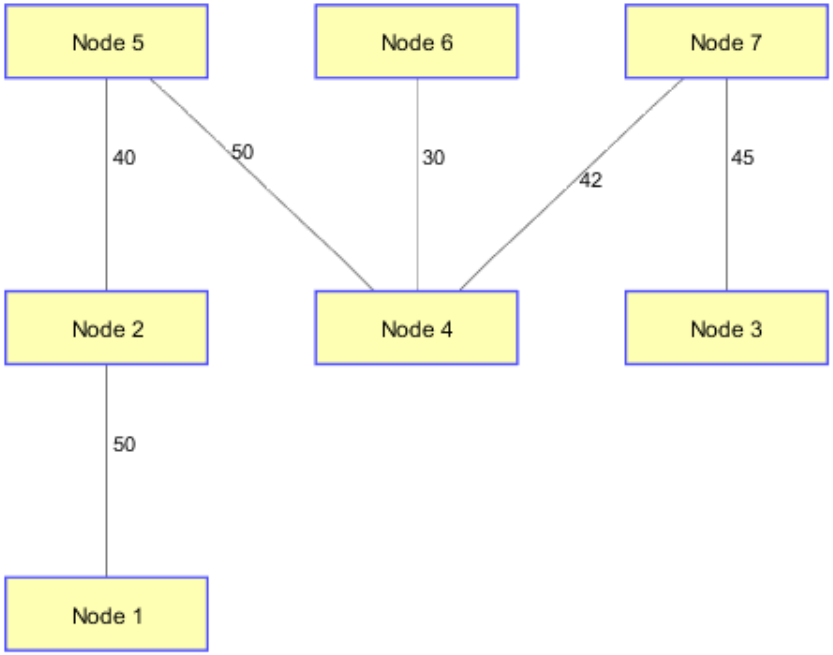

```
clc;clear;
G(1,2)=50;G(1,3)=60;
G(2,4)=65;G(2,5)=40;
G(3,4)=52;G(3,7)=45;
G(4,5)=50;G(4,6)=30;G(4,7)=42;
G(5,6)=70;G(6,6)=0;G(7,7)=0;
G=G+G';
G=sparse(G);
UG=tril(G);
view(biograph(UG,[],'ShowArrows','o
ff','ShowWeights','on'))
 [ST,pred] = graphminspantree(UG)

view(biograph(ST,[],'ShowArrows','o
ff','ShowWeights','on'))
```

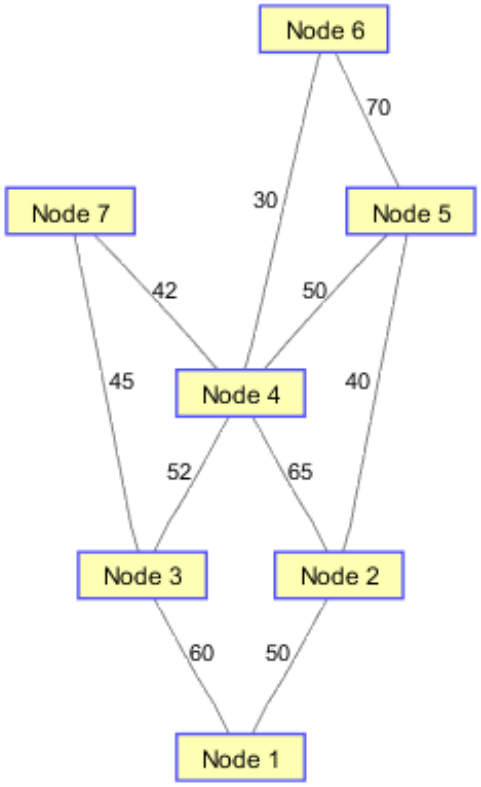
ST =

(2,1)	50
(5,2)	40
(7,3)	45
(5,4)	50
(6,4)	30
(7,4)	42

pred =

0 1 7 5 2 4 4

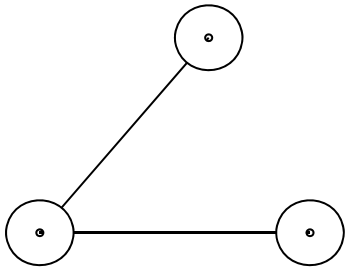


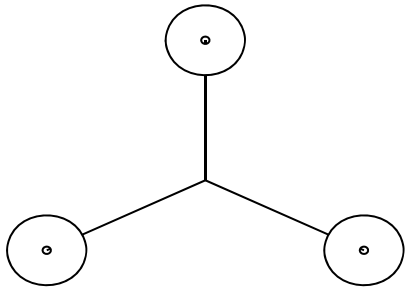


3.3 Steiner生成树

实际背景：在已有网络上选择连通几个城市的最廉价交通或通讯网。

数学模型：从已知的加权连通图上求取最小的树状子图，使此树包含指定的顶点子集。





第一个的边长为 $\sqrt{3}$ ，第二个的边长为1，总费用第二个更少。

分析：与传统的最小生成树相比，这里可以引入若干“虚拟站”并构造一个新的Steiner树，这样可以降低由一组站生成的传统的最小生成树所需的费用(降低的费用大概为13.4%)。而且为构造一个有n个顶点的网络的费用，最低的Steiner树决不需要多于(n－2)个虚设站。当然，有时最小Steiner生成树与最小生成树相同。寻求最小Steiner生成树的算法有Melzak算法(1961年)，

但是这是一个指数时间的算法，现在没有多项式时间的算法，换句话说它是一个NP问题。而且，这里的要求是用直折线代替欧氏直线距离，因而不能利用直接的算法。所以在解决这样的问题的时候，为减少运算的时间，理论上的分析是必要的：比如树的长度的下界，Steiner树的存在性，虚设站的位置等等。常用的算法还包括穷举法、模拟退火法等。

模拟退火法原理：模拟退火法(Simulated annealing, SA)是模拟热力学中经典粒子系统的降温过程，来求解极值问题。当孤立粒子系统的温度以足够慢的速度下降时，系统近似处于热力学平衡状态，最后系统将达到本身的最低能量状态，即基态，这相当于能量函数的全局极小点。其步骤如下(也称为Metropolis过程)：

（1） 给定初始温度 T_0，及初始点，计算该点的函数值 f(x)。

（2） 随机产生扰动 Δx，得到新点 x′=x+Δx，计算新点函数值 f(x′),及函数值差 Δf=f(x′)-f(x)。

（3） 若 Δf≤0，则接受新点，作为下一次模拟的初始点；

（4） 若 Δf>0，则计算新点接受概率：$p(\Delta f) = \exp(-\frac{\Delta f}{K \cdot T})$，产生 [0，1] 区间上均匀分布的伪随机数 r,r∈[0,1]，如果 p(Δf)≥r，则接受新点作为下一次模拟的初始点；否则放弃新点，仍取原来的点作为下一次模拟的初始点。

模拟退火法实例：

1、CMCM 97A题

97年全国大学生数模竞赛A题“零件的参数设计”，可以归结为非线性规划模型，由于目标函数很复杂，且又是一个多维函数，因此求解比较困难，为应用模拟退火法进行求解，将7个自变量的取值范围进行离散化，取步长为0.0001，这样，所有7个变量取值就组成了一个极为庞大的离散空间，而这个问题变成组合优化模型。

2、CMCM 98B题

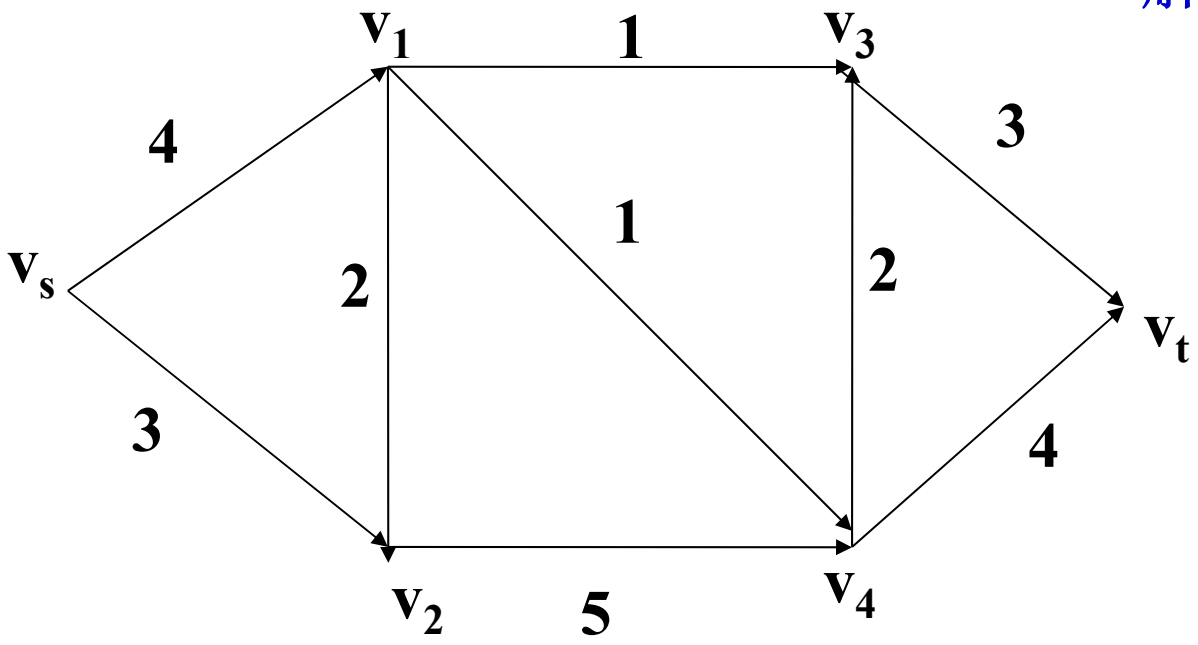
98年全国大学生数学建模竞赛B题“水灾巡视问题”，是一个推销员问题，本题有53个点，所有可能性大约为exp(53)，目前没有好方法求出精确解，既然求不出精确解，我们使用模拟退火法求出一个较优解，将所有结点编号为1到53，1到53的排列就是系统的结构，结构的变化规则是：从1到53的排列中随机选取一个子排列，将其反转或将其移至另一处，能量E自然是路径总长度。具体算法描述如下：

步1： 设定初始温度T，给定一个初始的巡视路线。
步2：步3 --8循环K次
步3：步 4--7循环M次
步4：随机选择路线的一段
步5：随机确定将选定的路线反转或移动，即两种调整方式：反转、移动。
步6：计算代价D，即调整前后的总路程的长度之差
步7：按照如下规则确定是否做调整：
如果D<0，则调整
如果D>0，则按照EXP(-D/T)的概率进行调整
步8：T*0.9-->T，降温

4.网络最大流问题

4.1 问题的提出

如下是一运输网络，弧上的数字表示每条弧上的容量，问：该网络的最大流量是多少？



解：设f_{ij}为弧(v_i, v_j)的流量。

$$\max f = f_{s1} + f_{s2}$$

$$\text{s.t.}\begin{cases} f_{s1} = f_{12} + f_{13} + f_{14} \\ f_{s2} + f_{12} = f_{24} \\ f_{13} + f_{43} = f_{3t} \\ f_{14} + f_{24} = f_{43} + f_{4t} \\ 0 \le f_{ij} \le c_{ij} \end{cases}$$

4.2 求解方法

4.2.1概念

1. 可行流和最大流

对于网络D＝(V，A，C)，网络D上的流 f 指定义在弧集合A上的一个函数 $f=\{f_{ij}\}$， f_{ij} 为弧 a_{ij} 的流量。满足下列条件的流 f 称为可行流：

- 容量限制条件：$0 \le f_{ij} \le c_{ij}$
- 平衡条件：$$\sum_{(v_i,v_j)\in A_i} f_{ij} - \sum_{(v_j,v_i)\in B_i} f_{ji} = \begin{cases} v(f) & i=s \\ 0 & i\neq s,t \\ -v(f) & i=t \end{cases}$$

A_i表示以节点v_i为起点的弧的集合，
B_i表示以节点v_i为终点的弧的集合。
$v(f)$称为可行流f的流量。

对于任何网络，其可行流 f 一定存在。如令 $f_{ij}=0$， f 为可行流，其 $v(f)=0$。

$v(f)$最大的可行流 f 称为<u>最大流</u>，记为 f^*。

可用以下LP模型求解。

$$\max Z = v(f)$$

$$s.t. \quad \sum_{(v_i,v_j)\in A_i} f_{ij} - \sum_{(v_j,v_i)\in B_i} f_{ji} = \begin{cases} v(f) & i = s \\ 0 & i \neq s,t \\ -v(f) & i = t \end{cases}$$

$$0 \le f_{ij} \le c_{ij}$$

$$0 \le f_{ji} \le c_{ji}$$

2. 零流弧和饱和弧

零流弧：$f_{ij}=0$的弧

饱和弧：$f_{ij}=c_{ij}$的弧

3. 前向弧和后向弧

若μ是网络中连接起点v_s和终点v_t的一条链，定义链的方向是从v_s到v_t，则链上的弧被分成两类：

前向弧：弧的方向与链的方向一致，μ^+表示μ中前向弧的集合

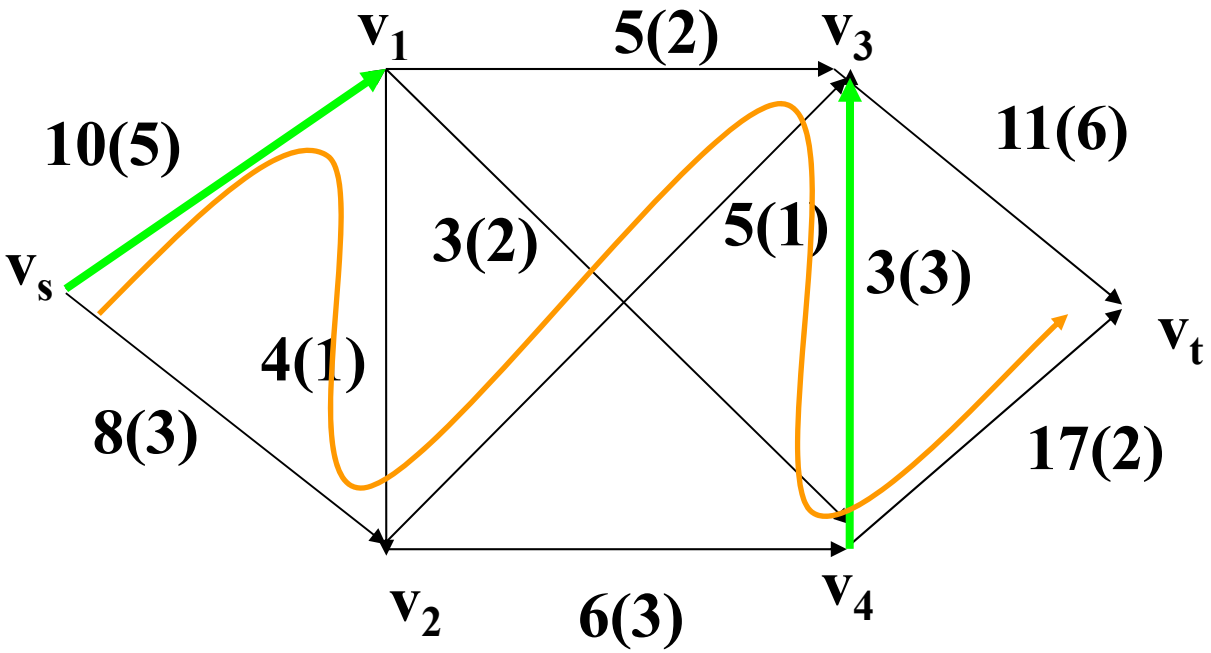
后向弧：弧的方向与链的方向相反，μ^-表示μ中后向弧的集合

4.2.2 增广路

若给一个可行流 $f=\{f_{ij}\}$，把网络中使 $f_{ij}=c_{ij}$ 的弧称为饱和弧，使 $f_{ij}<c_{ij}$ 的弧称为非饱和弧。把 $f_{ij}=0$ 的弧称为零流弧，$f_{ij}>0$ 的弧称为非零流弧。

若 μ 是网络中联结发点 v_s 和收点 v_t 的一条路，我们定义路的方向是从 v_s 到 v_t，则路上的弧被分为两类：一类是弧的方向与路的方向一致，叫做前向弧。前向弧的全体记为 μ^+。另一类弧与路的方向相反，称为后向弧。后向弧的全体记为 μ^-。

定义 设 f 是一个可行流，μ 是从 v_s 到 v_t 的一条路，若 μ 满足：前向弧是非饱和弧，后向弧是非零流弧，则称 μ 为（关于可行流 f）一条增广路。



在左图中，

$(v_s,v_1,v_2,v_3,v_4,v_t)$是一条增广链,因为$\mu^+$和$\mu^-$中的弧满足增广链的条件。如：

$(v_s,v_1)\in \mu^+, f_{s1}=5<10$

$(v_4,v_3)\in \mu^-, f_{43}=3>0$

5. 可扩充量

$$\theta = \min \begin{cases} c_{ij} - f_{ij} & (u_i, v_j) \in u^+ \\ f_{ij} & (u_i, v_j) \in u^- \end{cases}$$

6. 调整后的弧流量

$$f'_{ij} = \begin{cases} f_{ij} + \theta & (u_i, v_j) \in u^+ \\ f_{ij} - \theta & (u_i, v_j) \in u^- \\ f_{ij} & (u_i, v_j) \notin u \end{cases}$$

4.3 寻求最大流的标号法（Ford－Fulkerson）

思路：从一可行流出发，检查关于此流是否存在增广链。若存在增广链，则增大流量，使此链变为非增广链；这时再检查是非还有增广链，若还有，继续调整，直至不存在增广链为止。

从v_s到v_t的一个可行流出发（若网络中没有给定f，则可以设f是零流），经过标号过程与调整过程，即可求得从v_s到v_t的最大流。这两个过程的步骤分述如下。

（A）标号过程

在下面的算法中，每个顶点v_x的标号值有两个，v_x的第一个标号值表示在可能的增广路上，v_x的前驱顶点；v_x的第二个标号值记为δ_x，表示在可能的增广路上可以调整的流量。

（1）初始化，给发点v_s标号为$(0,\infty)$。

（2）若顶点v_x已经标号，则对v_x的所有未标号的邻接顶点v_y按以下规则标号

i）若$(v_x,v_y)\in A$，且$f_{xy}<c_{xy}$时，令$\delta_y=\min\{c_{xy}-f_{xy},\delta_x\}$则给顶点$v_y$标号为$(v_x,\delta_y)$，若$f_{xy}=c_{xy}$，则不给顶点$v_y$标号。

ii）$(v_y,v_x)\in A$，且$f_{yx}>0$，令$\delta_y=\min\{f_{yx},\delta_x\}$，则给$v_y$标号为$(-v_x,\delta_y)$，这里第一个标号值$-v_x$，表示在可能的增广路上，$(v_y,v_x)$为反向弧；若$f_{yx}=0$，则不给$v_y$标号。

（3）不断地重复步骤（2）直到收点v_t被标号，或不再有顶点可以标号为止。当v_t被标号时，表明存在一条从v_s到v_t的增广路，则转向增流过程（B）。如若v_t点不能被标号，且不存在其它可以标号的顶点时，表明不存在从v_s到v_t的增广路，算法结束，此时所获得的流就是最大流。

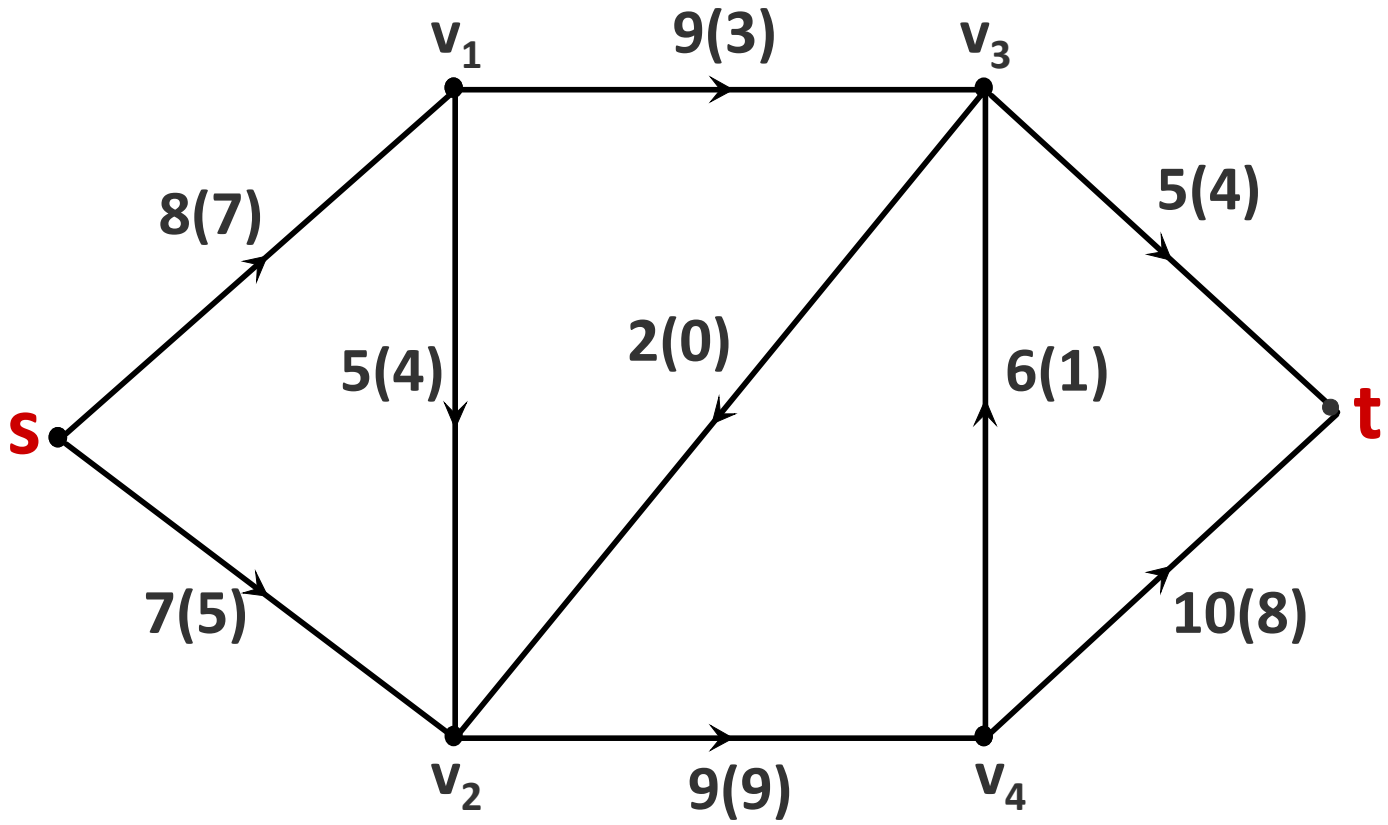
（B）增流过程

（1）令$v_y = v_t$。

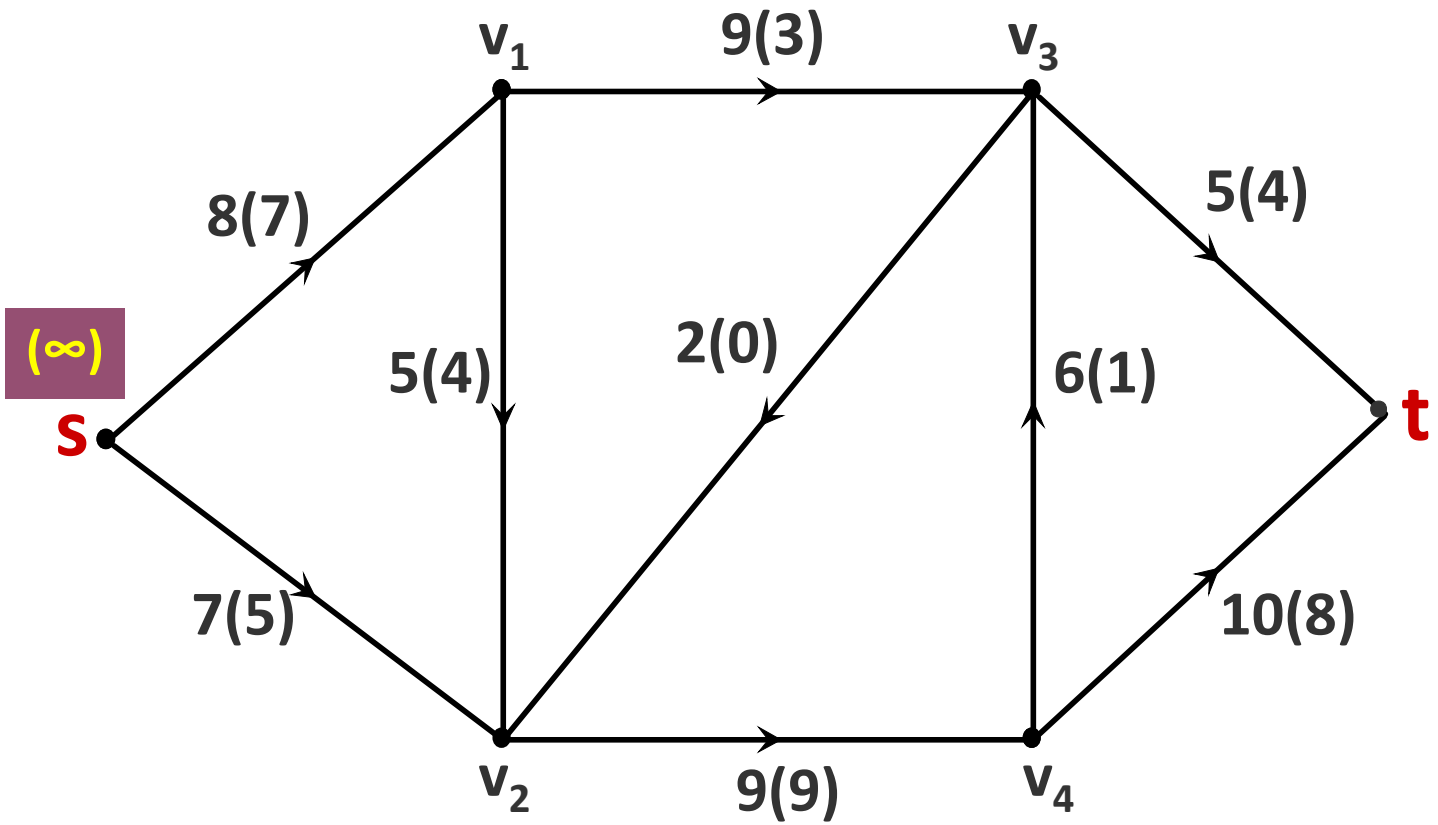
（2）若v_y的标号为(v_x, δ_t)，则 $f_{xy} = f_{xy} + \delta_t$；若$v_y$的标号为$(-v_x, \delta_t)$，则 $f_{yx} = f_{yx} - \delta_t$。

（3）若$v_y = v_s$，把全部标号去掉，并回到标号过程（A）。否则，令$v_y = v_x$，并回到增流过程（2）。

例6 用标号算法求下图中s→t的最大流量，并找出最小割。



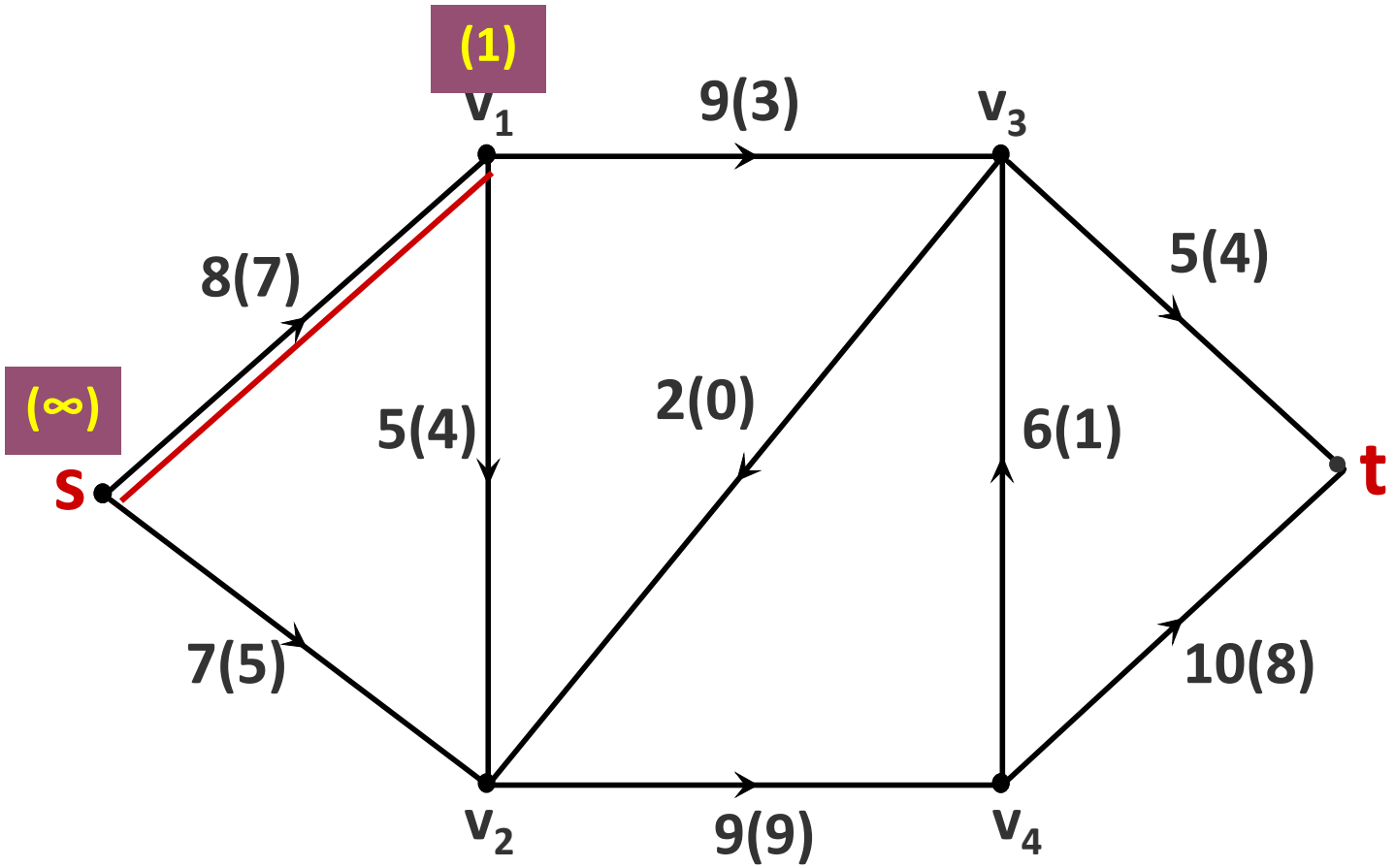
• 解：（1）先给s标号(∞)



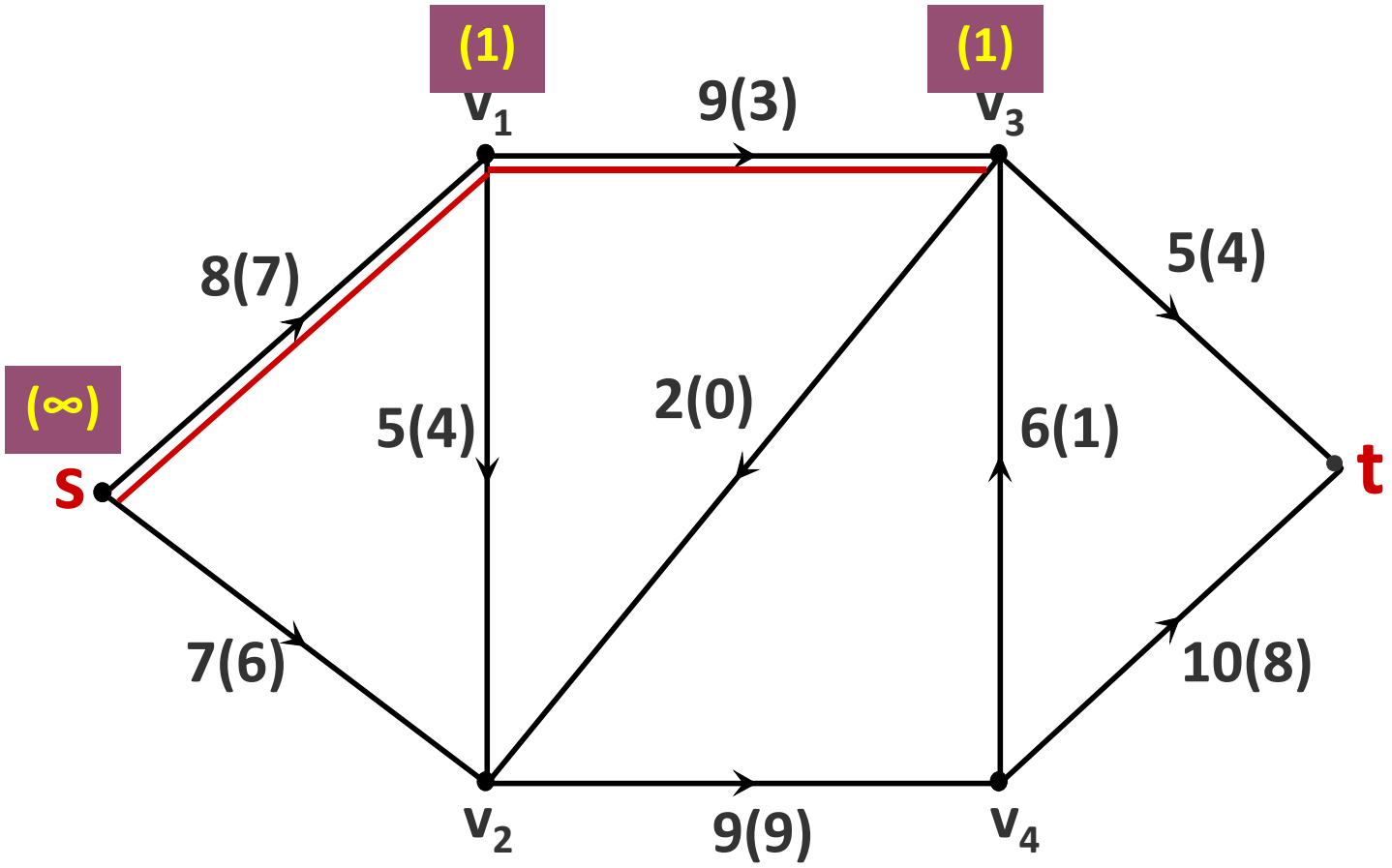
(2) 检查与s点相邻的未标号的点，因$f_{s1}<c_{s1}$，故对v_1标号

$=\min\{\infty,\ c_{s1}-f_{s1}\}=1$,

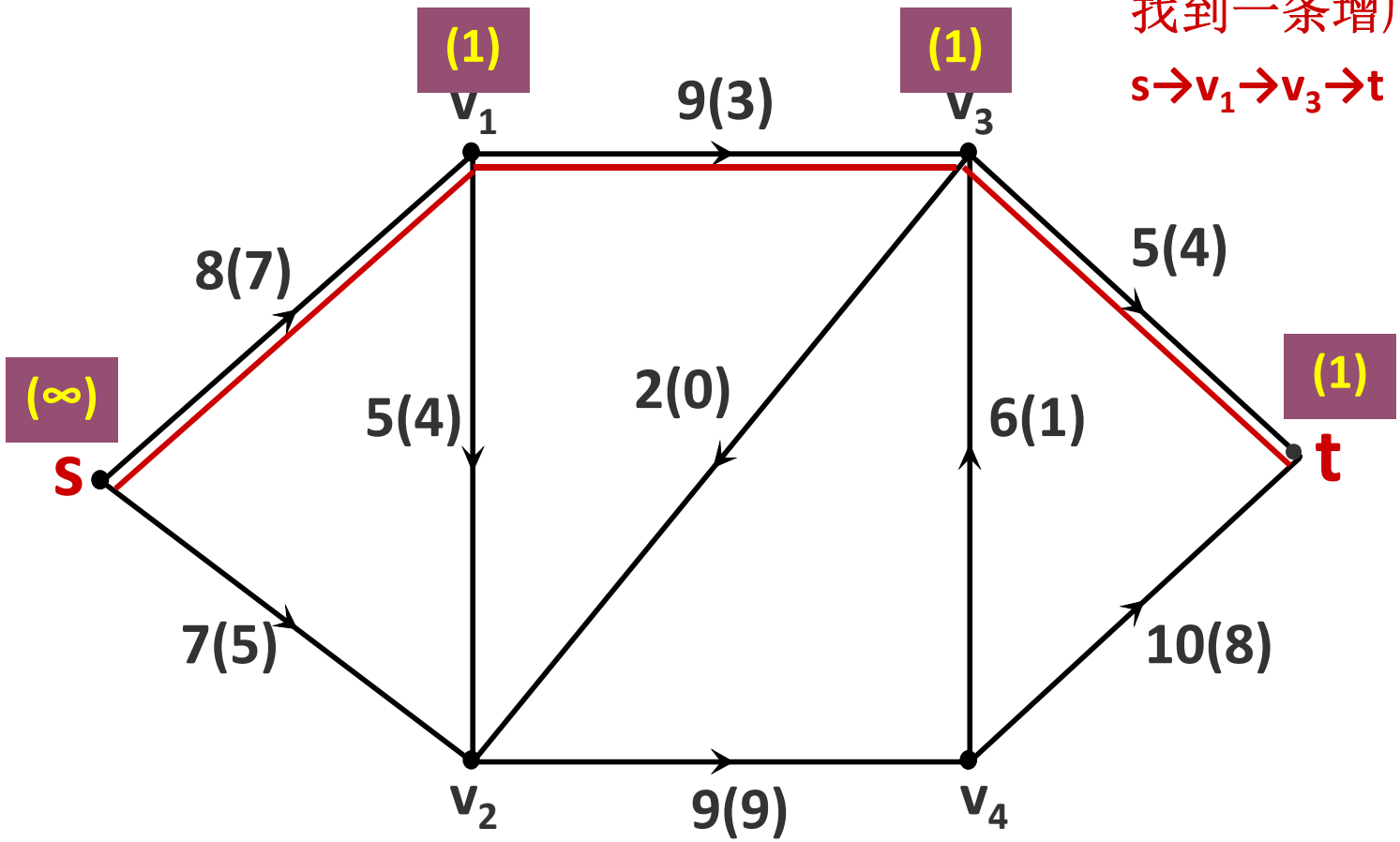
ε(1)



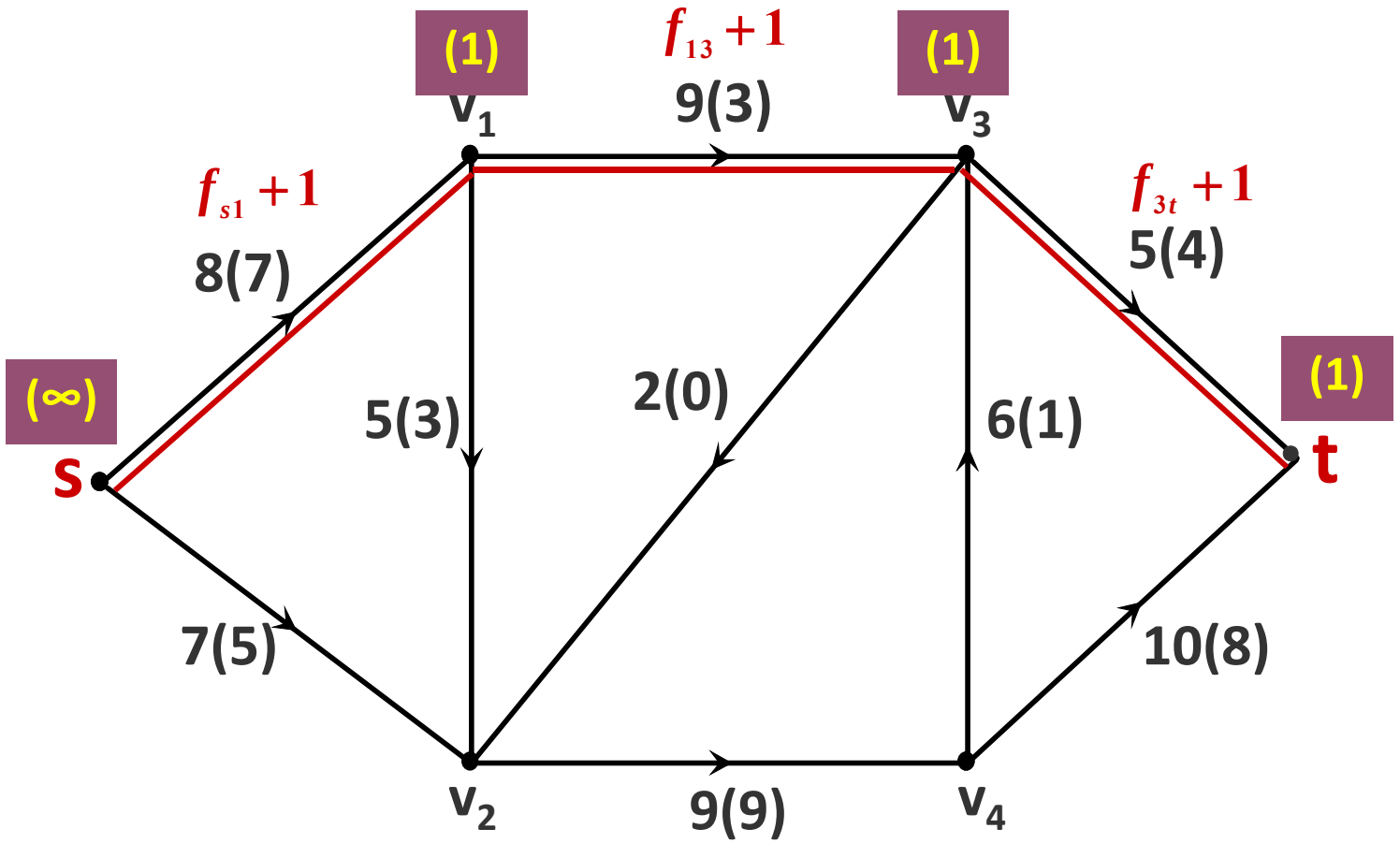
(2) 检查与v_1点相邻的未标号的点，因$f_{13}<c_{13}$，故对v_3标号$\varepsilon(3)$ $=\min\{1, c_{13}-f_{13}\}= \min\{1, 6\}= 1$



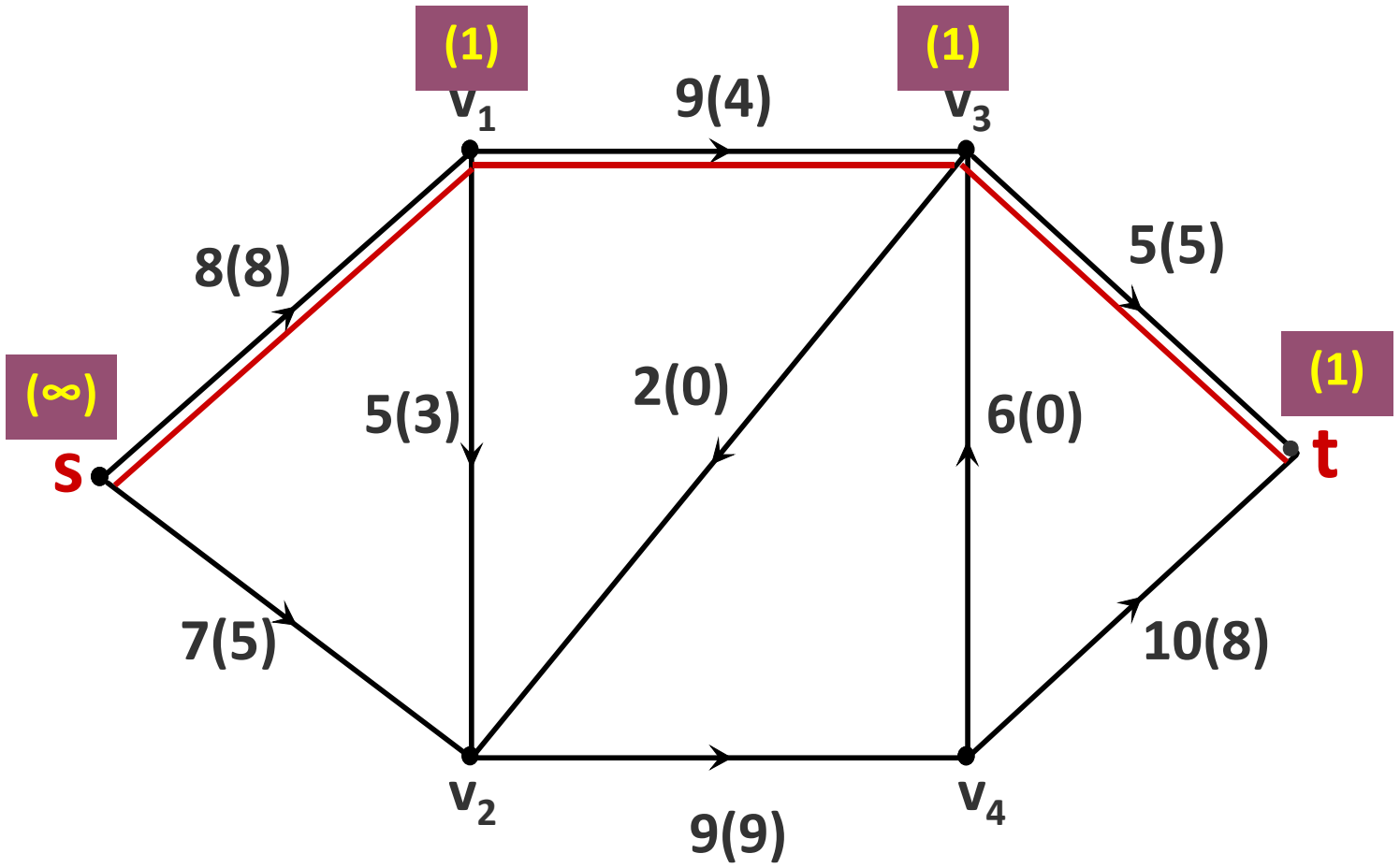
(3) 检查与v_3点相邻的未标号的点，因$f_{3t}<c_{3t}$，故对v_t标号 $\varepsilon(t)$ $=\min\{1, c_{3t}-f_{3t}\}= \min\{1, 1\}= 1$



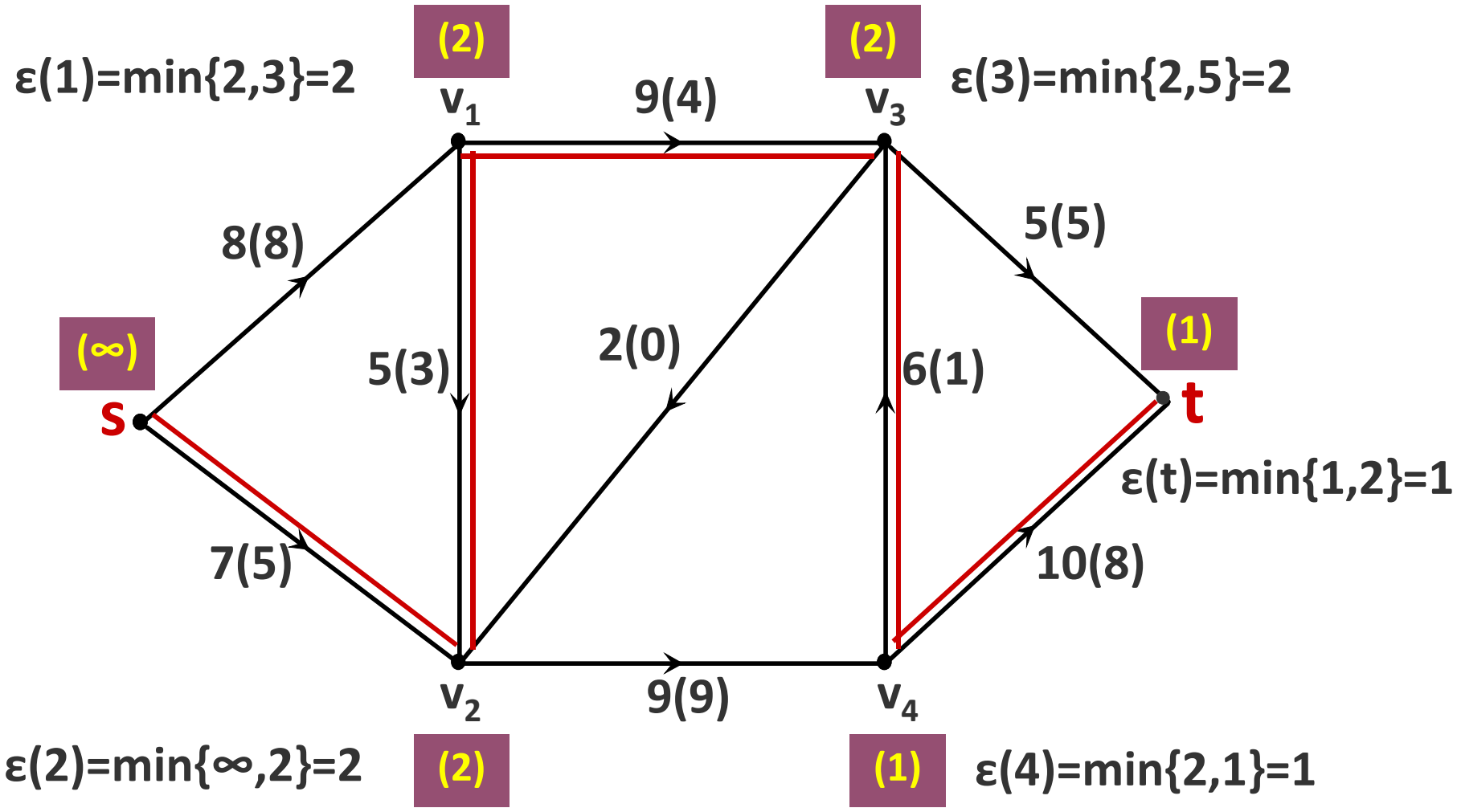
(4) 修改增广链上的流量，非增广链上的流量不变，得到新的可行流。



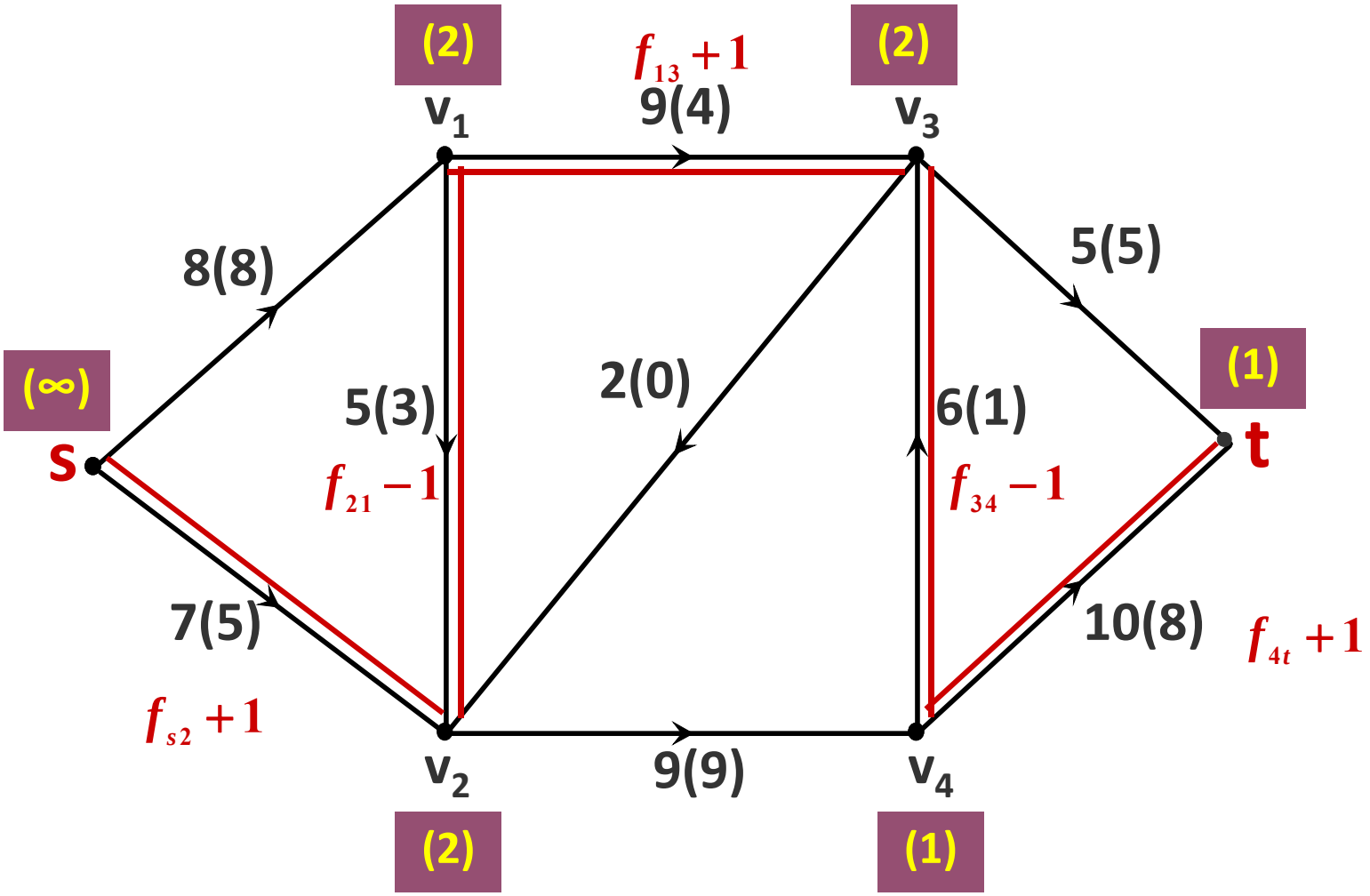
(5) 擦除所有标号，重复上述标号过程，寻找另外的增广链。



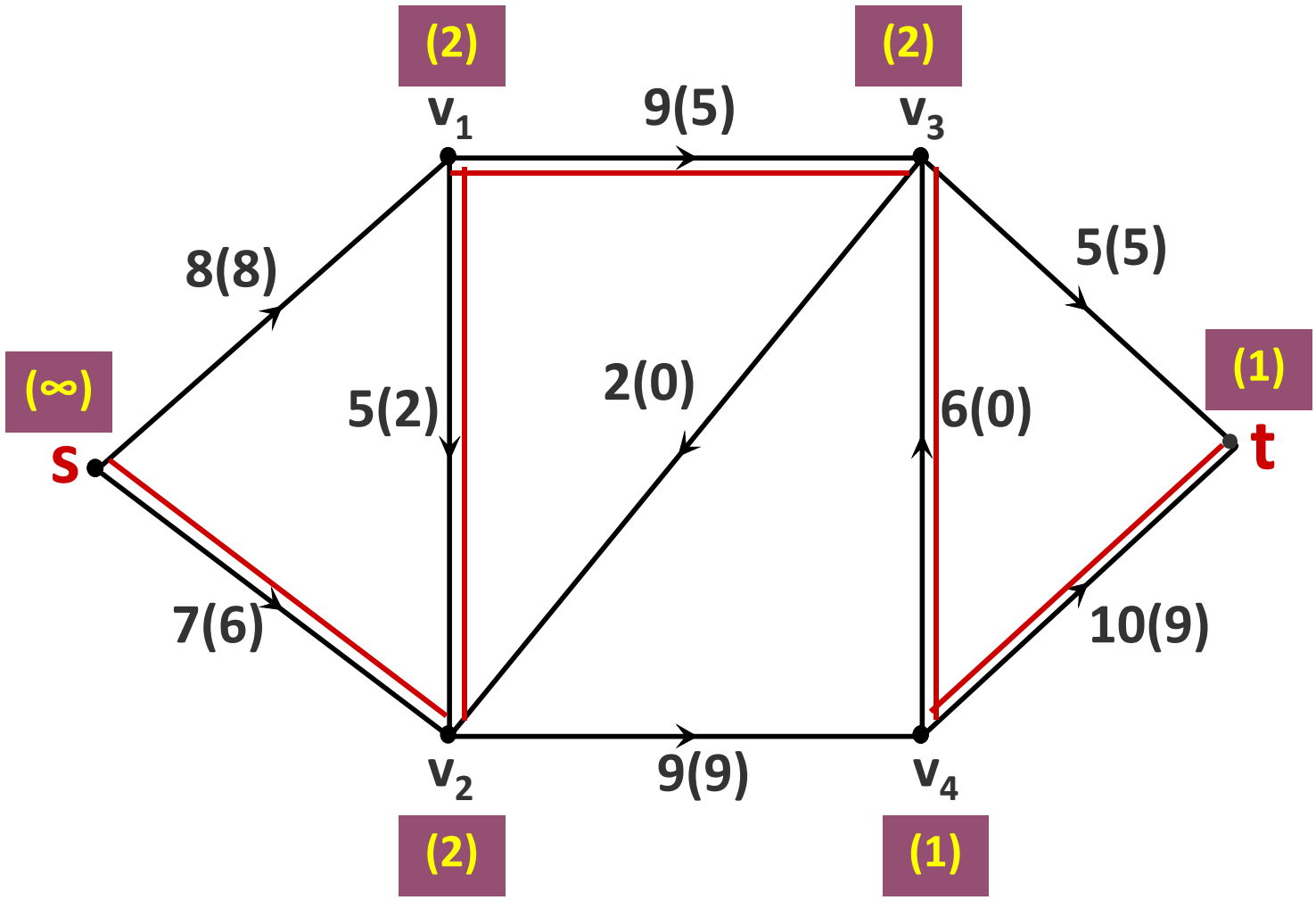
(5) 擦除所有标号，重复上述标号过程，寻找另外的增广链。



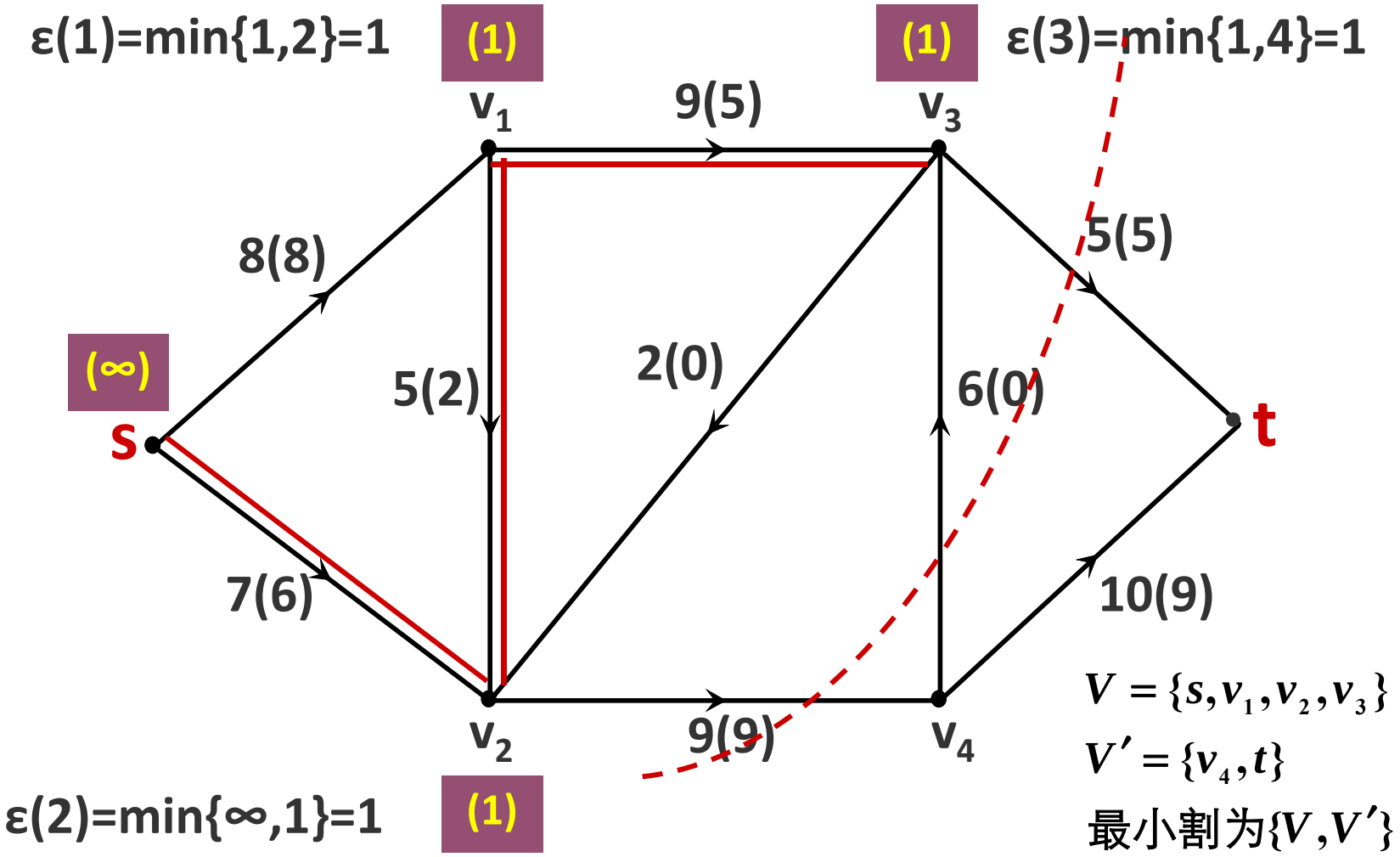
(6) 修改增广链上的流量，非增广链上的流量不变，得到新的可行流。



(7) 擦除所有标号，重复上述标号过程，寻找另外的增广链。

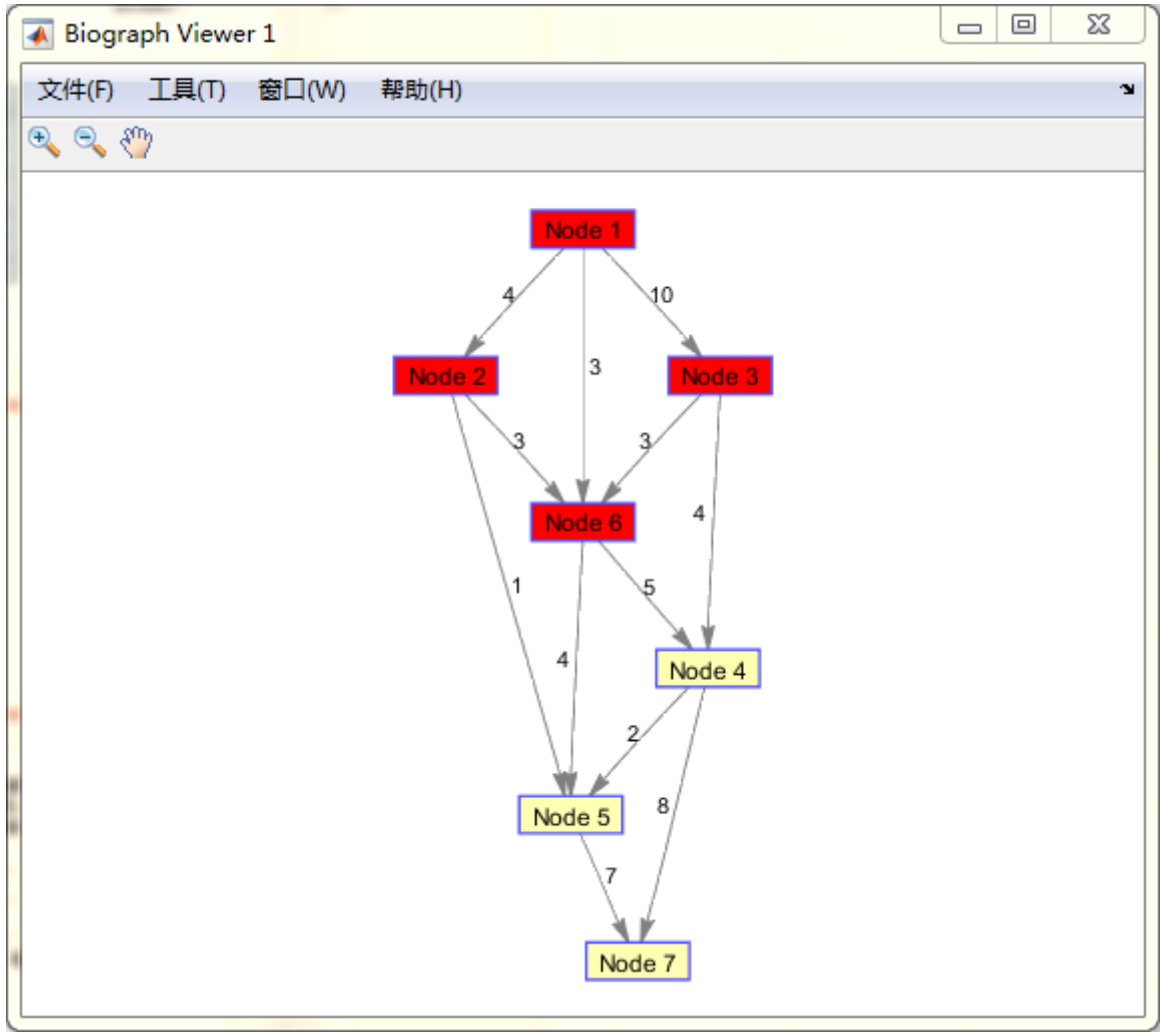


(7) 擦除所有标号，重复上述标号过程，寻找另外的增广链。

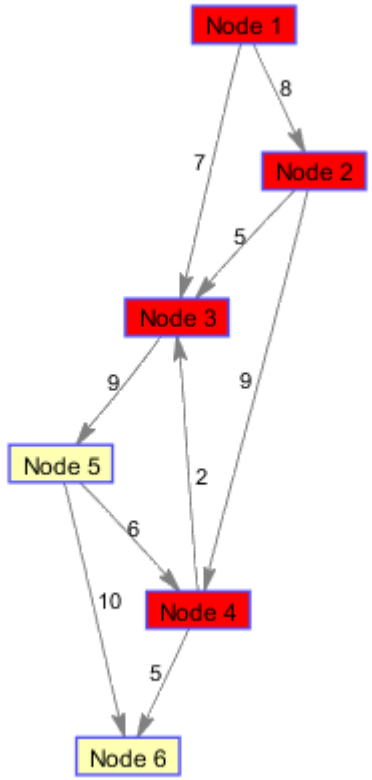


```
clc;clear;
G(1,2)=4;G(1,3)=10;G(1,6)=3;
G(2,6)=3;G(3,6)=3;G(2,5)=1;G
(6,5)=4;G(6,4)=5;G(3,4)=4;
G(4,5)=2;G(5,7)=7;G(4,7)=8;
G(7,7)=0;
G=sparse(G);
[MaxFlow, FlowMatrix, Cut] =
graphmaxflow(G, 1, 7)
h =
view(biograph(G,[],'ShowWeig
hts','on'))
view(biograph(FlowMatrix,[],
'ShowWeights','on'))
set(h.Nodes(Cut(1,:)),'Color
',[1 0 0])
```

```
MaxFlow =    14
FlowMatrix =
   (1,2)      4
   (1,3)      7
   (3,4)      4
   (6,4)      5
   (2,5)      1
   (4,5)      1
   (6,5)      4
   (1,6)      3
   (2,6)      3
   (3,6)      3
   (4,7)      8
   (5,7)      6
Cut =
     1     1     1     0     0     1     0
     1     1     1     0     0     0     0
     1     0     1     0     0     0     0
Biograph object with 7 nodes and 12 edges.
```


Biograph Viewer 1
文件(F)
工具(T)
窗口(W)
帮助(H)
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
4
10
3
3
3
4
1
5
4
2
8
7

```
G(1,2)=8;G(1,3)=7;
G(2,3)=5;G(2,4)=9;
G(3,5)=9;
G(4,3)=2;G(4,6)=5;
G(5,4)=6;G(5,6)=10;
G(6,6)=0;
G=sparse(G);
[MaxFlow, FlowMatrix, Cut]
= graphmaxflow(G, 1, 6)
h =
view(biograph(G,[],'ShowWei
ghts','on'))
view(biograph(FlowMatrix,[]
,'ShowWeights','on'))
set(h.Nodes(Cut(1,:)),'Colo
r',[1 0 0])
```



MaxFlow = 14

FlowMatrix =

(1,2)	8.0000
(1,3)	6.0000
(2,3)	1.0000
(4,3)	2.0000
(2,4)	7.0000
(3,5)	9.0000
(4,6)	5.0000
(5,6)	9.0000

Cut =

1 1 1 1 0 0

Biograph object with 6 nodes and 9 edges.

例　求图中从①到⑧的最大流。

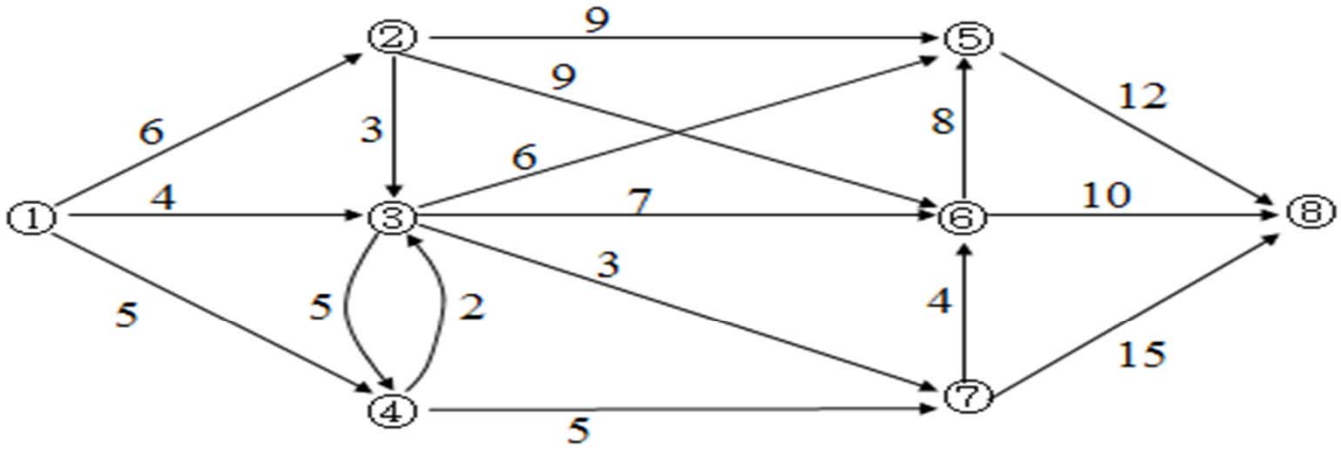


图 最大流问题的网络图

解　Matlab 图论工具箱求解最大流的命令，只能解决权重都为正值，且两个顶点之间不能有两条弧的问题。图 4.10 中顶点 3，4 之间有两条弧，为此，在顶点 4 和顶点 3 之间加入一个虚拟的顶点 9，并添加两条弧，删除顶点 4 到顶点 3 的权重为 2 的弧，加入的两条弧的容量都是 2。

5 最小费用最大流问题

在涉及“流”的问题时，人们考虑的不仅仅是流量，而且还有费用问题。例如在运输网络中，从源s到汇t所经历的路程，往往因为交通工具不同或道路本身结构不同而使各段路程运输费用不同。这时问题就变成了不仅要求s到t的运输量最大，而且要求这种运输方案的总费用最小。这类问题就属于最小费用最大流问题。

在给定网络$N=(V, A, C)$中，对每条弧$(v_i, v_j) \in A$，除给定弧容量c_{ij}外，还给出单位流量的费用$b_{ij} \geqslant 0$。f是N中的网络流，其总费用 $b(f)=\sum b_{ij} f_{ij}$ 。因此，把求最大流f，且使总费用最小的问题称为最小费用最大流问题，其数学模型为：

$$b^*(f)=\min \sum_{(v_i,v_j)\in A} b_{ij} f_{ij}$$

- 从上一节可知，求最大流的方法是从某个可行流出发，找到关于这个流的一个增广链 μ，沿着 μ 调整 f，对新的可行流试图再寻找它的增广链，如此反复直到找到最大流。现在要寻找最小费用最大流，我们首先考察一下，当沿着一条关于可行流 f 的增广链 μ，以 θ 调整 f，得到新的可行流 f' 时，显然新的可行流的 f' 费用大于 f 的费用，两者的差值我们称为增广链 μ 的费用。
- 可以证明，若 f 是流量为 $v(f)$ 的所有可行流中费用最小者，而 μ 是关于 f 的所有增广链中费用最小的增广链，那么沿 μ 去调整 f 得到的新的可行流 f' 一定是流量为 $v(f')$ 的所有可行流中的最小费用流。

- 由于$f=0$必定是流量为0的最小费用流，所以总可以从$f=0$开始。一般地，设已知f是流量为$v(f)$的最小费用流，余下的问题就是如何寻找关于f的最小费用增广链。为此我们构造有向赋权图$W(f)$，它的顶点是原网络N的顶点，而把N中的每一条弧(v_i, v_j)变成两个方向相反的弧(v_i, v_j)和(v_j, v_i)。定义$W(f)$中弧的权w_{ij}为

$$\omega_{ij}=\begin{cases} b_{ij} & f_{ij}<c_{ij} \\ +\infty & f_{ij}=c_{ij} \end{cases}$$

$$\omega_{ji}=\begin{cases} -b_{ij} & f_{ij}>0 \\ +\infty & f_{ij}=0 \end{cases}$$

- 于是在网络N中寻找关于f的最小费用增广链就等价于在赋权有向图$W(f)$中寻找从源到汇的最短路。

- 下面介绍求解这类问题的方法。
- 求最小费用最大流问题的基本思路是：从零流（$f_0=0$）的费用有向图 $W(f_0)$开始，用求最短路径的方法求出由s到t的最小费用链 μ_0，并对 μ_0按上节中标号法的调整方法进行流量的调整，所以新的网络流f_1必是最小费用的网络流。如果$v(f_1)=v$（v是给定的流量），则计算终止。否则，重新构造关于f的费用有向图$W(f_1)$，继续在$W(f_1)$上求出由s到t的最小费用链 μ_1，并在 μ_1上进行流量的调整，如此下去，直到求出流量最大的网络流为止。由此可见，求最小费用最大流的方法就是求最短路和求最大流方法的综合。

以下是最小费用最大流算法—“最小费用增广链法”。

算法思路

已知N=(V,A,C)，求最小费用最大流。

（1）开始，从一已知的最小费用网络流出发（也可以从零流（$f\equiv 0$）出发）。在N中找出关于f的最小费用增广链P。

●若P不存在，则当前的f即所求的最小费用最大流；转（3）；

●若存在，则转入调整过程；（转（2））。

（2）在P上对f进行调整，得$f^{(1)}$；

$$f_{ij}^{(1)}=\begin{cases} f_{ij}+\theta & a\in P^{+} \\ f_{ij}-\theta & a\in P^{-} \\ f_{ij} & a\notin P \end{cases}$$

其中调整量θ≥0，$f^{(1)}$为流量等于$V_f+\theta$的新的最小费用网络流，重复（1），在N中找关于$f^{(1)}$的最小费用增广链P_1。

（3）停标。f即为所求最小费用最大流。

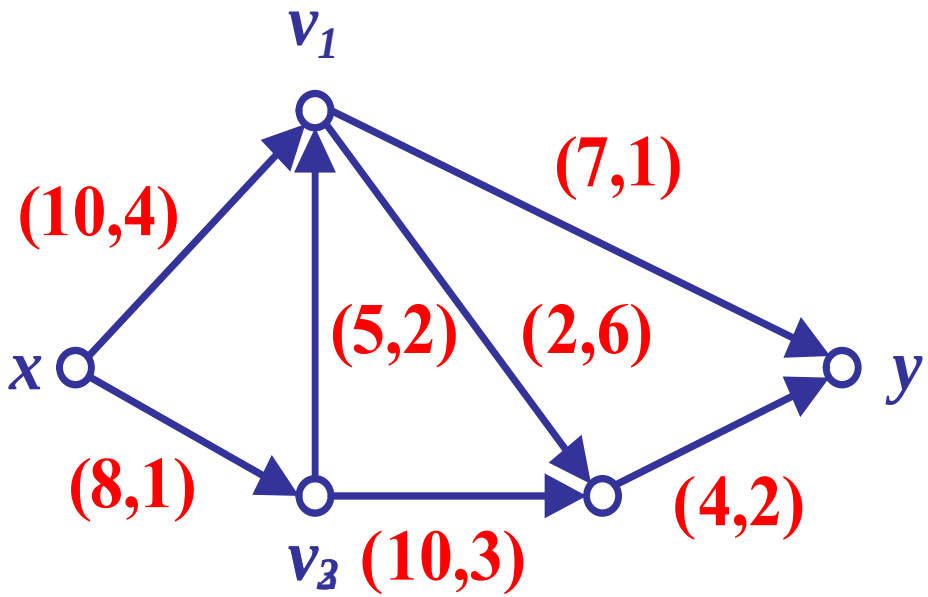
这里问题的关键是如何在N中找关于最小费用网络流f的最小费用增广链？—构造辅助赋权网络有向图$W(f)$。

2）辅助赋权有向网络$W(f)$的构造方法

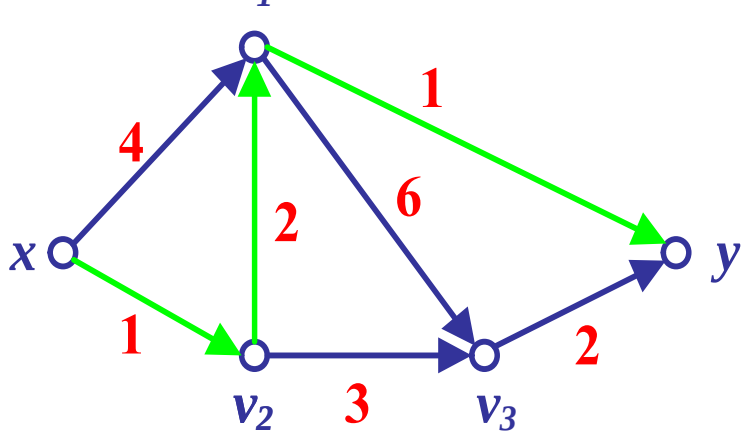
- 构造$W(f)$的目的是为了在N中找到关于最小费用网络流的，从s到t的最小费用增广链。构造$W(f)$时，总体上是要将N中所有可能作为关于f的增广链的弧全部集中在$W(f)$中，并同时为它们赋权（弧费用b_{ij}），具体方法如下：
- 设$W(f)=(V',A',\Omega)$，其中的V'，A'，Ω分表是网络$W(f)$的点集，弧集及弧权集合，它与N(V,A,C)的关系是：
- （1）顶点集：$V'(L)=V(N)$，即$W(f)$的顶点即原有的网络顶点。
- （2）弧集A'及其方向和赋权方法：可将N中弧分为以下3类。
- ①A集中的零流弧（即$f_{ij}=0$，方向$v_i \rightarrow v_j$）：这类弧只能作关于f增广链P中的前向弧，因此在A'集中，弧(v_i,v_j)的方向与原N中相同，赋权$+b_{ij}$。

- ②A集中的饱和弧（即$f_{ij}=c_{ij}$，方向$v_i \to v_j$）：这类弧只能作关于f增广链P的反向弧，因此在A’集中，弧(v_i,v_j)的方向与原N中相反，赋权$-b_{ij}$。
- ③A集中的非饱和、非零流弧（即$0<f_{ij}<c_{ij}$，方向$v_i \to v_j$）：这类弧即可以作为关于f增广链P中的前向弧，又可作反向弧，因此在A’集中，点v_i，v_j之间，应该有正反两个方向的弧同时存在，其中方向与原N中相同的弧(v_i,v_j)赋$+b_{ij}$，而与原N中方向的弧(v_i,v_j)赋权$-b_{ij}$。
- 这样，在N中找到关于f的从s到t的最小费用增广链，就等价于在赋权有向网络$W(f)$中，找到从s到t的最短路径。

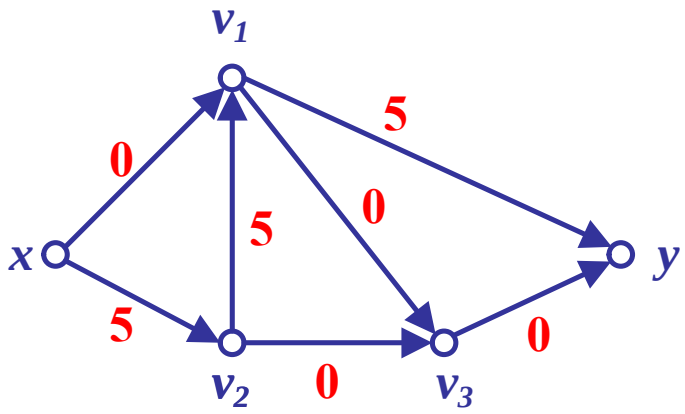
- 应用举例
- 　　最小费用最大流算法应用非常广泛，如运输问题、生产计划问题和指派问题等一类决策问题，均可以转化为网络模型的最小费用最大流问题进行求解。
- 　　例 已知一交通网络如下图，要把10单位的货物从x运到y去，弧旁边的数字（c_{ij},b_{ij}），c_{ij}代表每条线路的最大运输能力，b_{ij}代表通过该线路单位物资的运价。考虑：如何调运能使费用最省?



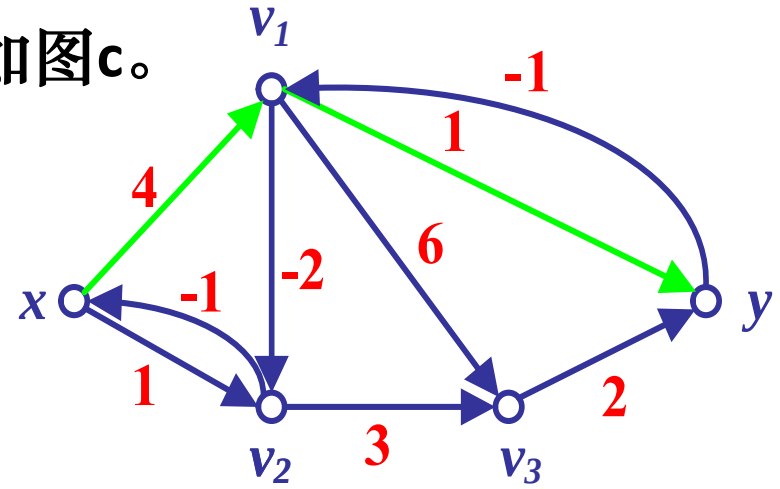
- 解 这是一个求流量为10单位的最小费用流问题。
- （1）$f^0=0$为初始网络流，作相应的费用有向图$W(f^0)$，如下图。



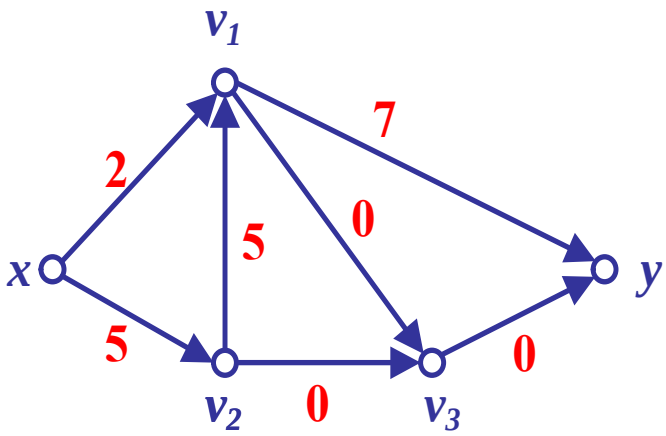
- （2）在$W(f^0)$上用*Dijkstra*标号法求出从x到y的最短路径（最小费用链）$P_0=(x,v_2,v_1,y)$，如上图所示，并按$\vartheta=min(8,5,7)=5$进行流量的调整，得到新的网络流f_1的流量有向图如下图。



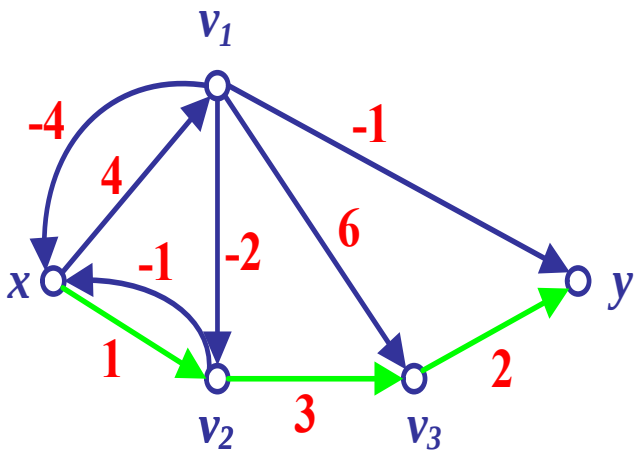
- （3）在$D(f^0)$和$P_0=\{x,v_2,v_1,N\}$的基础上，重新构造网络流f_1相应的费用有向图$D(f_1)$，由于$(x,v_2)\in P_0^+$，$f_{x2}=5<c_{x2}$，因此，保留原费用方向，并添一反向费用弧，$b_{2x}=-1$；$(v_2,v_1)\in P_0^+$，$f_{21}=5=c_{21}$，因此，将原弧的方向改为相反方向，且$b_{12}=-2$；$(v_1,y)\in P_0^+$，$f_{1y}=5<c_{1y}$，因此，保留原费用方向，并添一反向费用弧，$b_{1y}=-1$；其余弧不变，如图c。

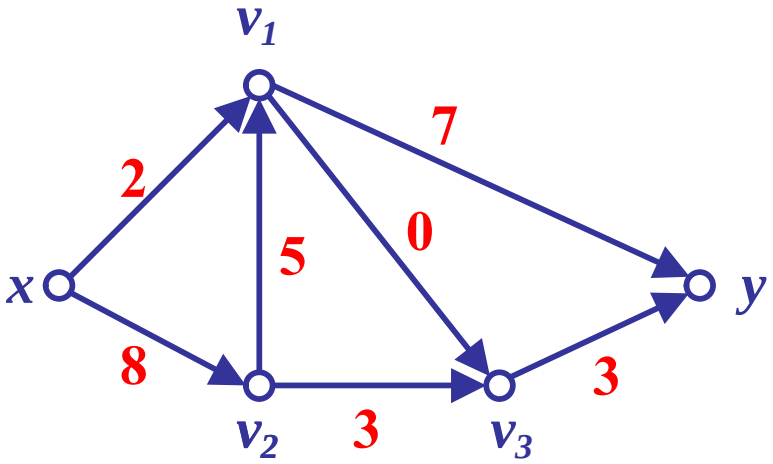


- （4）在$D(f_1)$上，由于边上有负权，所以求最N路径不能用Dijkstra编号法，可用逐次逼近法。求出由x到y的最短路径（最小费用链）$P_1=\{x,v_1,y\}$，对P_1进行流量的调整，可得网络流f_2如下图所示。



- （5）用同样的方法重新构造$D(f_2)$，如下图所示，求出x到y的最短路径（最小费用链）$P_2=\{x,v_2,N_3,y\}$，并对P_2进行流量的调整，的网络流f_3。
- 由于$v(f_3)=10$，所以计算终止。此时，最小费用为$b^*(f)=2\times4+8\times1+5\times2+7\times1+3\times3+2=48$。





最小费用最大流问题可以归结为两个线性规划问题，用线性规划模型求出最大流量$v(f_{\max})$，以及用如下的线性规划模型求出最大流对应的最小费用。

$$\begin{aligned} &\min \sum_{(v_i,v_j)\in A} b_{ij} f_{ij}, \\ &\text{s.t.} \quad 0 \le f_{ij} \le c_{ij}, \quad \forall (v_i,v_j) \in A, \\ &\sum_{j:(v_i,v_j)\in A} f_{ij} - \sum_{j:(v_j,v_i)\in A} f_{ji} = d_i, \end{aligned} \tag{4.2}$$

其中

$$d_i = \begin{cases} v(f_{\max}), & i = s, \\ -v(f_{\max}), & i = t, \\ 0, & i \ne s,t. \end{cases}$$

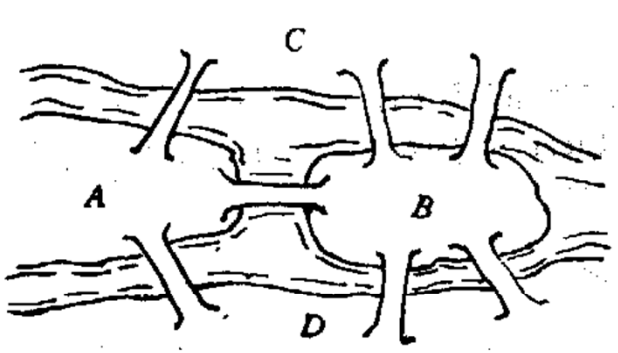
这里$v(f_{\max})$表示(4.1)线性规划模型求得的最大流的流量。

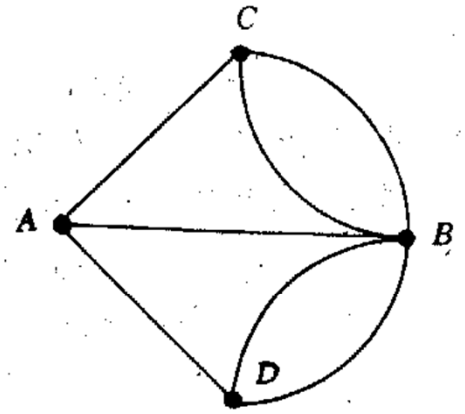
6 欧拉图与中国邮递员问题

七桥问题的数学模型：

用A、B表示两座小岛，C、D表示两岸，

连线AB表示A、B之间有一座桥。





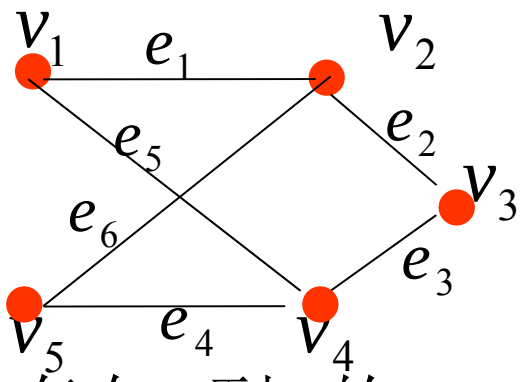
问题简化为：

在该图中，从任一点出发，能否通过每条线段一次且仅仅一次后又回到原来的出发点

结论：不存在这样一种走法。

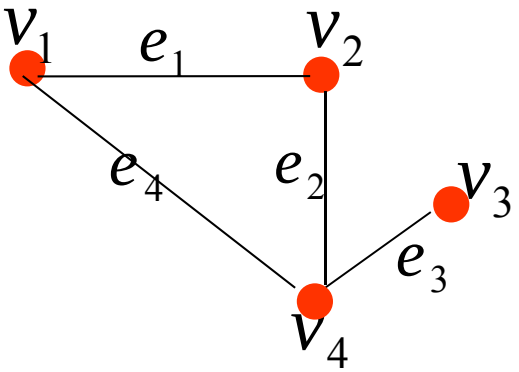
一笔画问题

- 欧拉链：给定一个连通多重图G，若存在一条链，过每边一次，且仅一次，则称这条链为欧拉链。
- 欧拉圈若存在一个简单圈，过每边一次，且仅一次，则称这个圈为欧拉圈。
- 欧拉图：一个图若有欧拉圈，则称欧拉图。



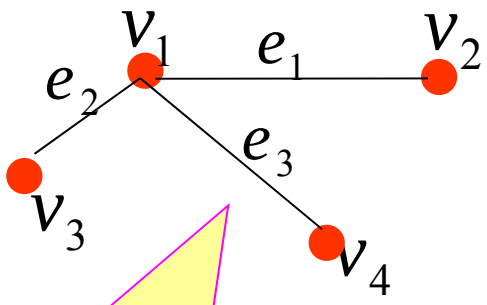
存在v_2到v_4的一条欧拉链：

$\{v_2,e_1,v_1,e_5,v_4,e_3,v_3,e_2,v_2,e_6,v_5,e_4,v_4\}$



存在v_3到v_4的一条欧拉链：

$\{v_3,e_3,v_4,e_2,v_2,e_1,v_1,e_4,v_4\}$



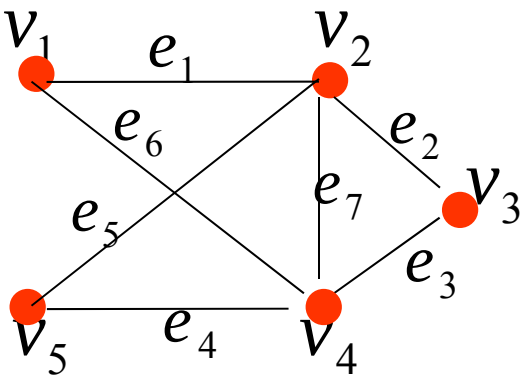
任两点之间都不存在欧拉链

●欧拉回路：

欧拉图

圈

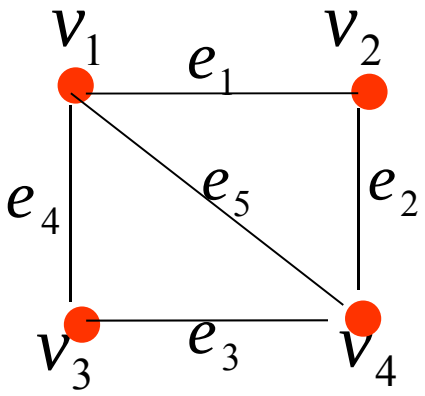
设G是一个无向连通图，若 存在一个回路，经过 G中的每一条边一次且仅一次 ，则称这个回路为欧拉 回路



存在欧拉回路：

$\{v_1,e_1,v_2,e_2,v_3,e_3,v_4,e_7,v_2,e_5,v_5,e_4,v_4,e_6,v_1\}$

该图特点：$d(v_i)$均为偶数

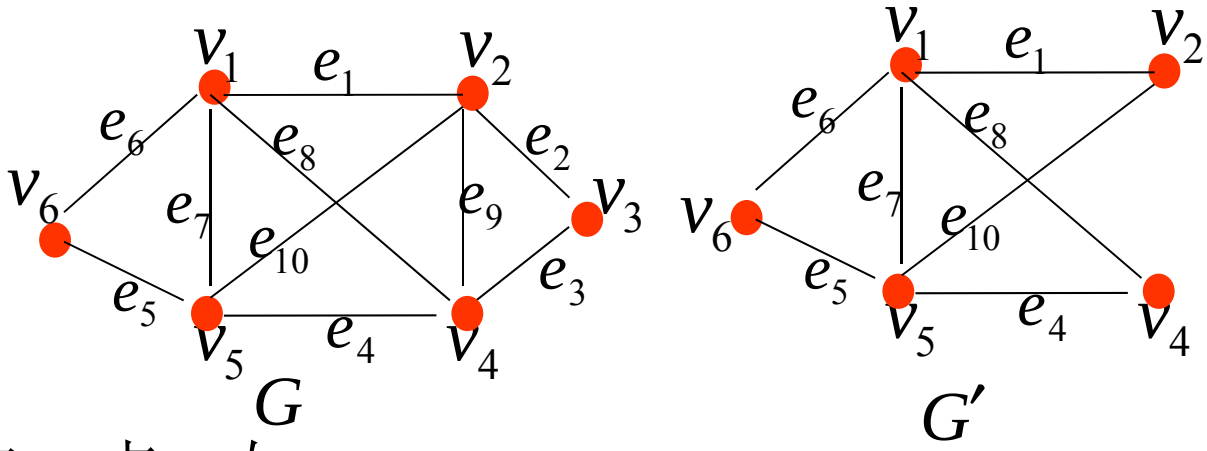


该图不存在欧拉回路

存在奇点

定理　无向连通图G为欧拉图的充要条件是G中无奇点

例：G为连通图，$d(v_i)$为偶数，求一欧拉回路。



任取一点，如 v_3，找一个以 v_3 为起点的一个简单回路 C_1

简单回路C_1:$\{v_3, e_3, v_4, e_9, v_2, e_2, v_3\}$

记$G' = G - C_1 = (V', E')$，$E' = E - E_1$，V'是E'中边的端点

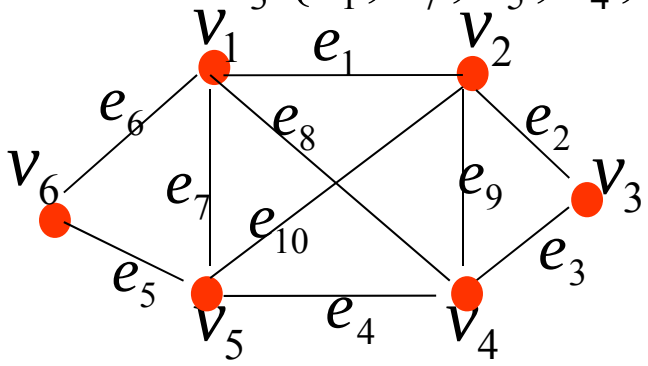
在G'中，以G'与C_1的公共顶点v_2为起点取一个简单回路C_2

简单回路C_2:$\{v_2, e_{10}, v_5, e_5, v_6, e_6, v_1, e_1, v_2\}$

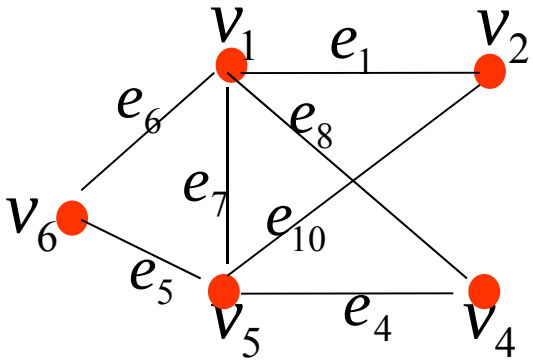
记$G''=G'-C_2=(V'',\ E'')$，$E''=E'-E_1$，V''是E''中边的端点

在G''中，以G''与C_2的公共顶点v_1为起点取一个简单回路 C_3

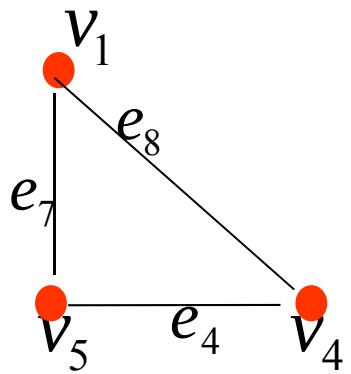
简单回路C_3:$\{v_1,e_7,v_5,e_4,v_4,e_8,v_1\}$



G



G'



G''

简单回路C_1:$\{v_3,e_3,v_4,e_9,v_2,e_2,v_3\}$

简单回路C_2:$\{v_2,e_{10},v_5,e_5,v_6,e_6,v_1,e_1,v_2\}$

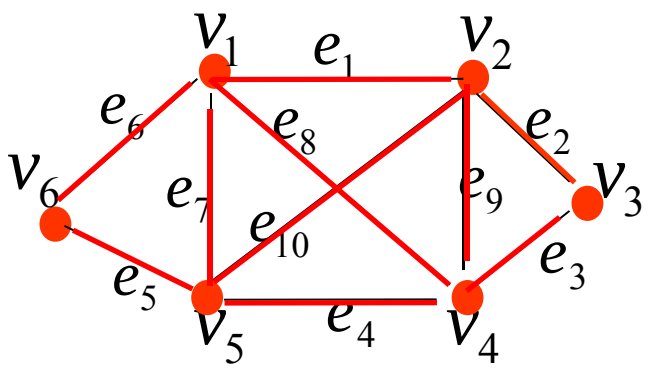
简单回路C_3:$\{v_1,e_7,v_5,e_4,v_4,e_8,v_1\}$

把C_3从C_2的v_1点处插入 C_2，得一简单回路 $\overline{C}$，

$\overline{C}$:$\{v_2, e_{10}, v_5, e_5, v_6, e_6, v_1, e_7, v_5, e_4, v_4, e_8, v_1, e_1, v_2\}$

再把$\overline{C}$从C_1的v_2点处插入 C_1，即得所求欧拉回路：

$$\{v_3, e_3, v_4, e_9, v_2, e_{10}, v_5, e_5, v_6, e_6, v_1, e_7, v_5, e_4, v_4, e_8, v_1, e_1, v_2, e_2, v_3\}$$



G

推论 无向连通图G有欧拉路的充要条件是G中恰有两个奇点。

一笔画问题：

- 一个连通图的顶点都是偶点，没有奇点，则该图可以一笔画出（从任一点出发均可）；
- 一个连通图的顶点恰有两个奇点，其余均为偶点，则从任一奇点出发，可以一笔画出该图，而终点则是另一个奇点；
- 一个连通图的顶点有两个以上的奇点，则该图不能一笔画出。

寻找欧拉圈或欧拉链可以通过Fleury算法或Hierholzer算法.

Fleury算法步骤如下：

①任取v_0属于V（G），令$P_0 = v_0$；
②设$P_i = v_0 e_1 v_1 e_2 \ldots\ldots e_i v_i$, 如果$E(G)-\{e_1, e_2, \ldots\ldots e_i\}$中没有与$v_i$关联的边，则计算停止；否则按下述的条件从$E(G)-\{e_1, e_2, \ldots\ldots e_i$中任取一条边$e_{i+1}$；
（a） e_{i+1}与v_i相关联。
（b）除非无别的边可供选择，否则e_{i+1}不应该选择$G_i = G-\{e_1, e_2, \ldots\ldots e_i\}$中的桥。
设e_{i+1}=（ v_i， v_{i+1}），把e_{i+1} v_{i+1}加入P_i，
③令$i = i+1$，返回②.

Hierholzer算法步骤如下：

设G是欧拉图，

1) 任选顶点为v_0，以v_0为起点，生成一个闭道路T_0, i←0；

2) 在T_i上选择一个顶点v_i，要求v_i有不在T_i上的关联边；在图G-E(T_i)中构造以vi为起点的闭道路T’,将T’插入到T_i的顶点vi处，形成一个更长的闭道路$T_{i+1}=T_i \cup T'$；

3）若E(T_{i+1})=E(G), 则停止；否则i ←i+1, 返回2)。

停止后，T_{i+1}即是所求欧拉回路。

Mathematica命令

FindEulerianCycle

FindEulerianCycle [g]
求图 g 中的欧拉圈.

FindEulerianCycle [g, k]
求至多 k 个欧拉圈.

FindEulerianCycle [{ $v \to w$, ... }, ...]
使用规则 $v \to w$ 指定图 g.

中国邮路问题

提出问题的人：管梅谷教授

时间：1962年

提出的问题：

一个邮递员从邮局出发分送邮件，要走完他负责投递的所有街道，最后再返回邮局，应如何选择投递路线，才能使所走的路线最短？

邮路问题的图论描述：

取一无向赋权连通图G=（V，E）

E中的每一条边对应一条街道

每条边的非负权l(e)=街道的长度

V中某一个顶点为邮局，其余为街道的交叉点

在G上找一个圈，该圈过每边至少一次，且圈上所有边的权和最小

邮路问题的图论描述：

在无向连通赋权$G=(V,E)$上找一个圈，

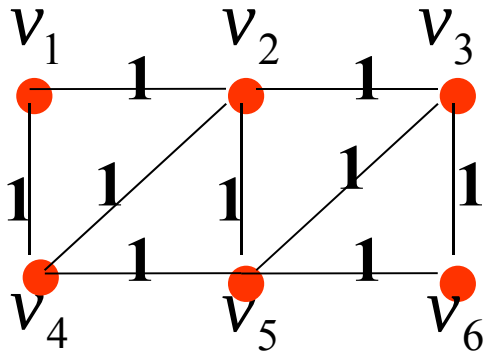
该圈过每边至少一次，且圈上所有边的权和最小

1、若G中的顶点均为偶点，即G中存在欧拉回路，则该回路过每条边一次且仅一次，此回路即为所求的投递路线

2、G中有奇点：不存在欧拉回路

投递路线：至少有一街道要重复走一次或多次

例：设G为邮路图，其中v_1为邮局



$\because$ $d(v_3), d(v_4)$为奇数 ，G不是欧拉图

$\therefore$ G不存在欧拉回路，

即不存在每条街道走一次且只走一次的投递路线

路线 C_1： $v_1v_2v_3v_6v_5v_4v_1v_2v_3v_5v_2v_4v_1$ 权和 $=12$

路线 C_2： $v_1v_4v_2v_4v_5v_3v_6v_5v_2v_3v_2v_1$ 权和 $=11$

路线 C_2优于路线 C_1

重复边

分析： 路线C_1重复走过边： (v_1, v_2) ， (v_2, v_3) ， (v_1, v_4)

重复边的权和 $=3$

路线C_2重复走过边： (v_2, v_4) ， (v_2, v_3)

重复边的权和 $=2$

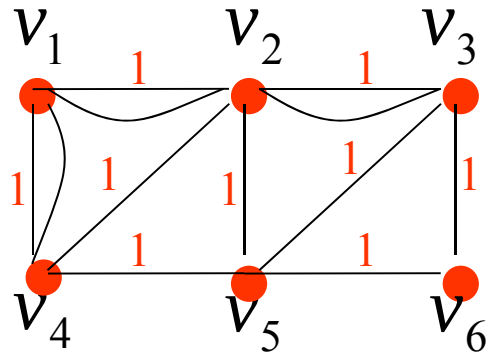
结论：

选择最佳投递路线=选择重复边的权和最小的路线

对路线 C_1：$v_1v_2v_3v_6v_5v_4v_1v_2v_3v_5v_2v_4v_1$

重复边：(v_1, v_2)，(v_2, v_3)，(v_1, v_4)

G_1为欧拉图，且C_1为G_1的一条欧拉回路

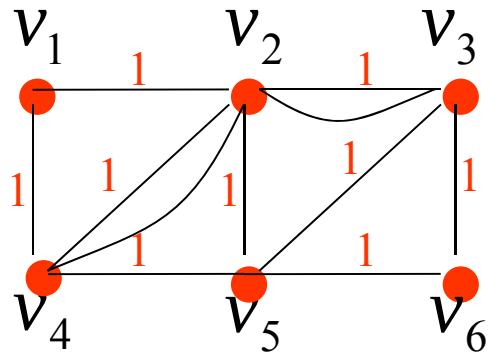


G_1

对路线 C_2：$v_1v_4v_2v_4v_5v_3v_6v_5v_2v_3v_2v_1$

重复边：(v_2, v_4)，(v_2, v_3)

G_2为欧拉图，且C_2为G_2的一条欧拉回路



G_2

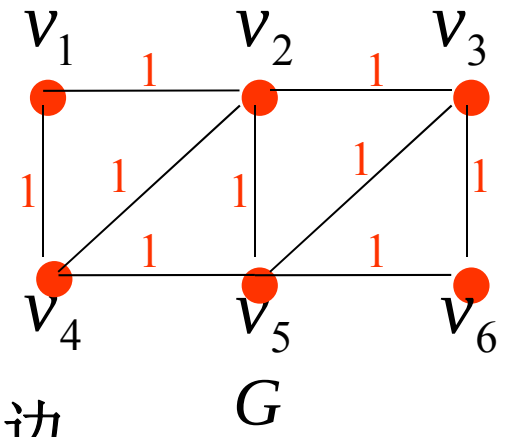
→ 一条投递路线对应一个欧拉图

且投递路线为该图的一条欧拉回路

反之，对邮路图**G**，

任取奇点v_4到v_3的一条链

如：v_4, v_5, v_6, v_3



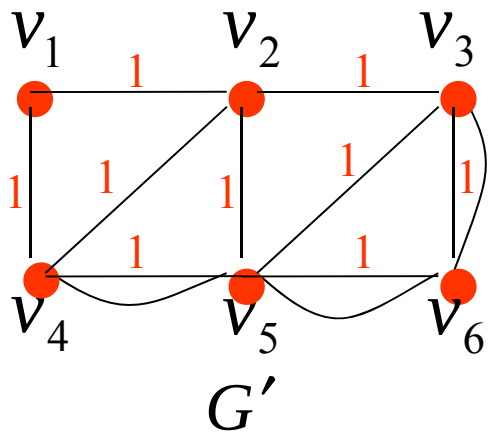
G

对该链上的每一条边增加一条重复边

G'为欧拉图 ,即G'存在欧拉回路

即G'对应一条投递路线 C

C：$v_1v_2v_3v_6v_3v_5v_6v_5v_2v_4v_5v_4v_1$



G'

投递路线 ⟷ 欧拉图

结论：对任意一个含奇点的邮路图G，由于奇点的个数为偶数个，把每两个配成一对，由于G为连通图，每对奇点之间至少存在一条链，对该条链上的每一条边增加一条重复边，可得一欧拉图，该欧拉图对应一条投递路线

寻找最佳投递路线方法：

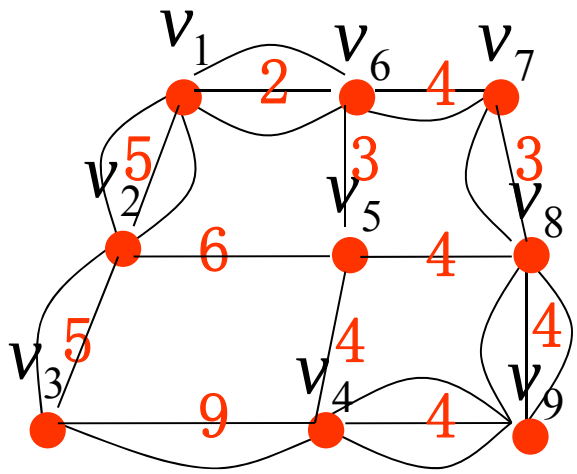
在原邮路图上增加一些重复边得一个欧拉图，，在所得欧拉图上找出一条欧拉回路。计算重复边的权和，重复边权和最小欧拉回路既为所求的最佳投递路线

管梅谷——奇偶点图上作业法

- 在奇点间加边，构造初始可行方案。
- 寻找改进可行方案：在两奇点间检查所有链，若某链的长度小于已加重复边的长度，则在该链的每边加上重复边，去掉原重复边。
- 重复以上步骤，直到任意两奇点间加重复边的链是最短的为止。

奇偶点图上作业法：

例：求解右图所示的邮路问题 (v_1为邮局)



G

第一步：确定一个初始可行方案

方法：检查图G中是否有奇点

无奇点：　图G已是欧拉图，找出一条以v_1为起点的欧拉回路，该回路就是最佳投递路线

有奇点：　把所有奇点两两配成一对，每对奇点找一条链，在该条链上的每一条边增加一条重复边，得一个欧拉图G_1，由G_1所确定的欧拉回路即为一个可行方案

G中有奇点：　v_2，v_4，v_6，v_8

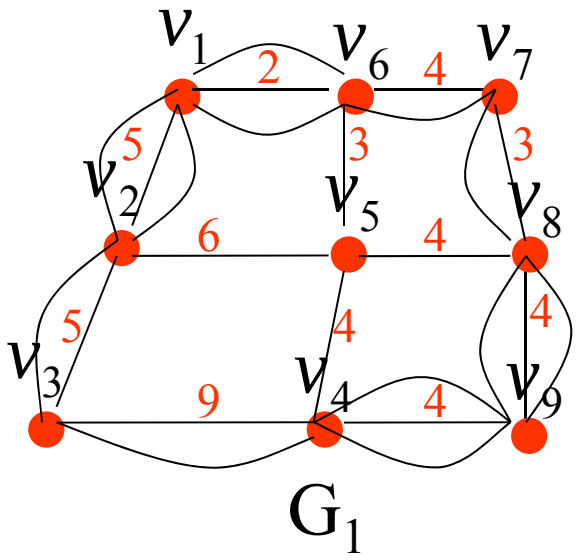
取v_2到v_4的一条链：　$v_2v_1v_6v_7v_8v_9v_4$

取v_6到v_8的一条链：　$v_6v_1v_2v_3v_4v_9v_8$

G_1是欧拉图，重复边权和= 51

显然G_1不是最佳方案

第二步：调整可行方案，使重复边权和下降



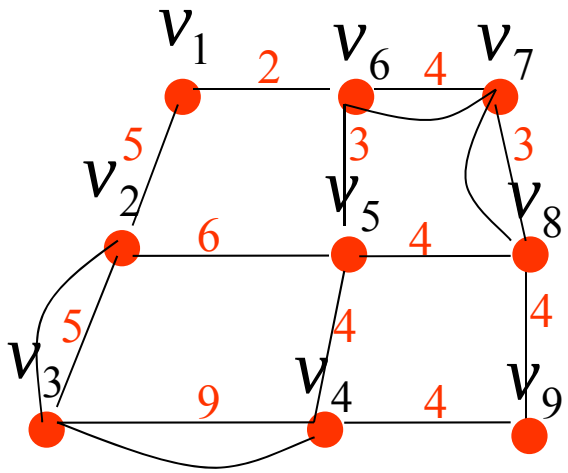
G_1

步骤1、 若图中某条边有两条或多于两条的重复边

同时去掉偶数条，使图中每一条边最多有一条重复边

可得到重复边权和较小的欧拉图 G_2

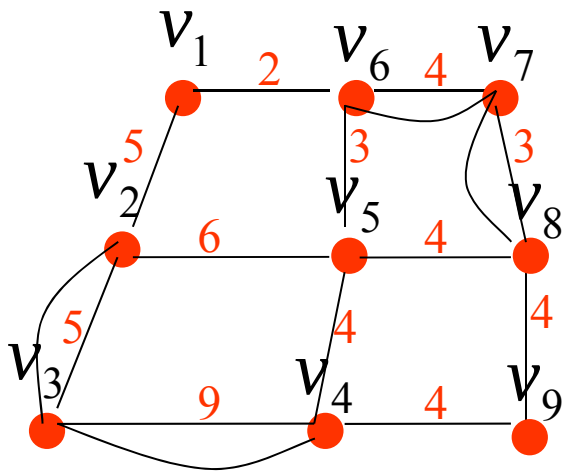
G_2的重复边权和= 21



G_2

G_2是欧拉图，

重复边权和=21



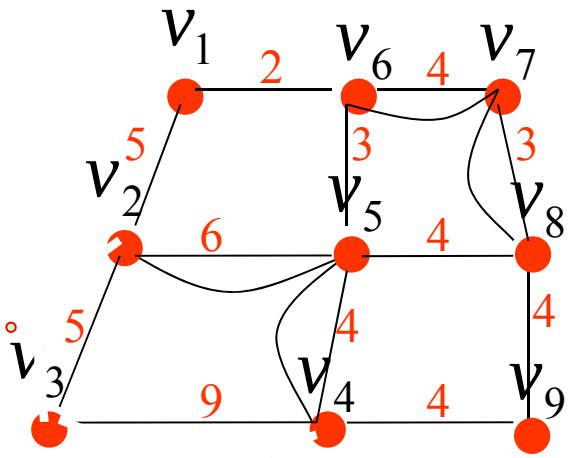
G_2

2、使图中每个初等圈重复边的权和不大于该圈权和的一半（困难）

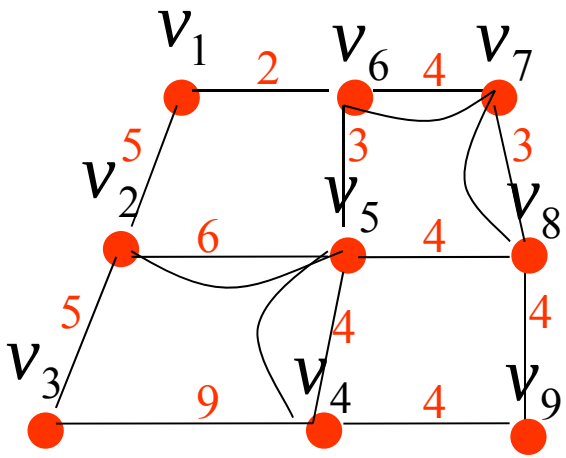
G_3是欧拉图，

重复边权和=17

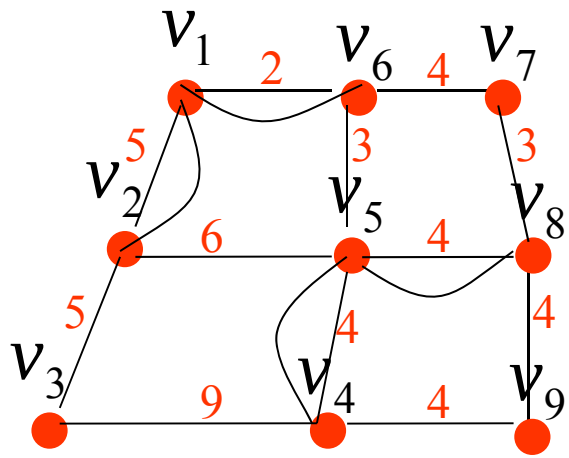
9个初等圈



G_3



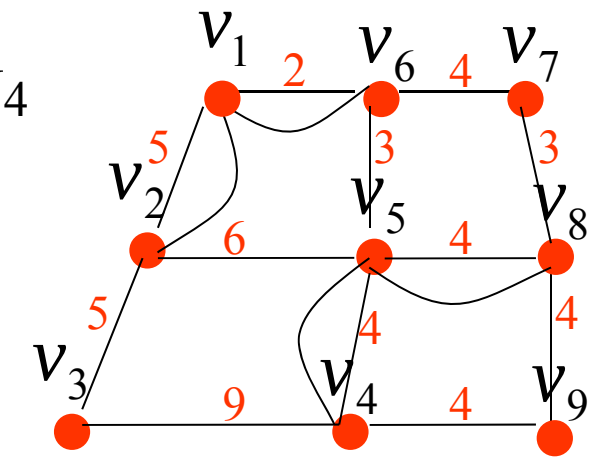
G_3



G_4

G_3的初等圈	权和	重复边权和	
（1）$v_1v_2v_5v_6v_1$	16	6	√
（2）$v_6v_5v_8v_7v_6$	14	7	√
（3）$v_2v_3v_4v_5v_2$	24	10	√
（4）$v_5v_4v_9v_8v_5$	16	4	√
（5）$v_1v_2v_5v_8v_7v_6v_1$	24	13	×

G_4



G_4是最佳方案

最佳投递路线：

$v_1v_2v_3v_4v_5v_2v_1v_6v_5v_4v_9v_8v_5v_8v_7v_6v_1$

G_4的初等圈	权和	重复边权和	
（1） $v_1v_2v_5v_6v_1$	16	7	√
（2） $v_6v_5v_8v_7v_6$	14	4	√
（3） $v_2v_3v_4v_5v_2$	24	4	√
（4） $v_5v_4v_9v_8v_5$	16	8	√
（5） $v_1v_2v_5v_8v_7v_6v_1$	24	11	√
（6） $v_2v_3v_4v_9v_8v_5v_2$	32	4	√
（7） $v_1v_2v_3v_4v_5v_6v_1$	28	11	√
（8） $v_6v_5v_4v_9v_8v_7v_6$	22	4	√
（9） $v_1v_2v_3v_4v_9v_8v_7v_6v_1$	36	7	√

奇偶点图上作业法：

第一步：确定一个初始可行方案

方法：检查图G中是否有奇点。

无奇点：图G已是欧拉图，找出一条以v_0为起点的欧拉回路，该回路就是最佳投递路线

有奇点：把所有奇点两两配成一对，每对奇点找一条链，在该条链上的每一条边增加一条重复边

第二步：调整可行方案，使重复边权和下降

1、使图中每一条边最多有一条重复边

若图中某条边有两条或多于两条的重复边，同时去掉偶数条

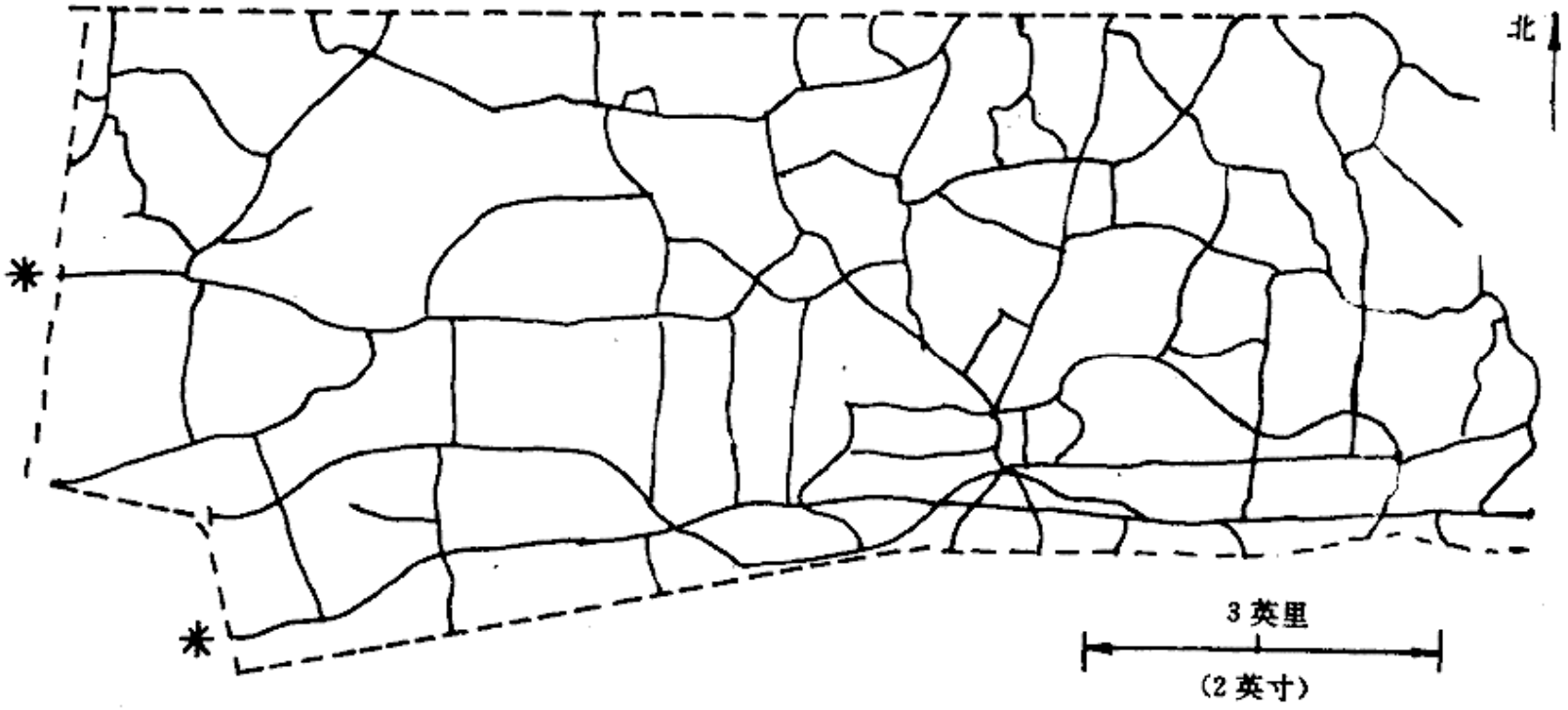
2、使图中每个初等圈重复边的权和≤该圈权和的一半

若图中某初等圈重复边的权和大于该圈权和的一半

去掉圈中的重复边同时将圈中没有重复边的边加上重复边

铲雪车的行驶路线问题（MCM　90B题）

地图中实线表示马里兰州(Maryland)咸克米克市(w5comico)清除积雪区域的双车道道路，虚线是州高速公路。雪后，两辆扫雪车从地图*号标出的两点的每一地点以西约4英里处出发清扫道路上的积雪。试为两车找出有效的路径。扫雪车可以通过高速公路进出市内道路。



- **7.旅行商问题Traveling Salesman Problem**

发展历史

- 旅行商问题，也称货郎担问题，是一个较古老的问题。其起源已经有些模糊了。最早大概可以追溯到 1759 年 Euler 提出的骑士旅行问题。

- 十九世纪初，爱尔兰数学家William R. Hamilton和英国数学家Thomas P. Kirkman研究过一些与旅行商问题相关的数学问题。

- 二十世纪初，人们开始研究通用形式的旅行商问题。
- 二十世纪二十年代，数学家和经济学家 **Karl Menger** 在维也纳向他的同事提出了这个问题。
- 二十世纪三十年代，旅行商问题出现在 **Princeton** 大学的数学圈子里，主要的推动者有 **Hassler Whitney** 与 **Merrill Flood**。
- 二十世纪四十年代，统计学家(**Mahalanobis(1940), Jessen(1942), Gosh(1948), Marks(1948)**)把它和农业应用联系在一起研究。美国**RAND** 公司也推动了这个问题的发展。
- 最终，旅行商问题成为了组合优化问题中的一个困难问题原型的典型代表。求解这种问题令人望而生畏：当问题规模变大的时候，路径的数目将是个天文数字，逐一检查它们几乎是不可能的。在很长的一段时间内，没有任何解决这个问题的好想法出现.

- 1954 年，旅行商问题的求解终于获得了突破。George Dantizig, Ray Fulkerson 和 Selmer Johnson 提出了一个求解旅行商问题的算法并用它成功地解决了一个有49 个城市的实例。这个规模在当时相当引人注目;
- 1977 年，Groetschel 找到了有 120 个城市的旅行商问题的最优路径;
- 1987年，Padberg 与 Rinaldi 找到了规模为 532 和 2392 的旅行商问题的最优路径；Groetschel与Holland找到了规模为666的旅行商问题的最优路径。
- Applegate, Bixby,Chavátal 和 Cook 于 1994 年,1998年和2001年解决了规模为 7397, 13509和 15112的旅行商问题。
- 2004 年，一个具有 24978 个城市的旅行商问题的最优路径由 Applegate, Bixby,Chavátal, Cook 和 Helsgaun 找到。这是到目前为止精确找到最优解的最大规模的旅行商问题.

- 旅行商问题吸引了越来越多的人对它进行研究。其中，有数学家，计算机科学家，运筹学家，还有一些其它领域的研究者。

- 然而，该问题是否存在一个有效的通用的求解方法仍然是一个开放性的问题。事实上，旅行商问题的解决将意味着**P=NP**问题的解决。**Clay Mathematics Institute** 曾悬赏 **100** 万美元来寻求这个问题的解法，但没人拿到这个奖。

- 旅行商问题的描述

  - 旅行商问题(TSP) 的文字描述可以表达如下：给定一组 N 个城市和它们两两之间的直达距离，找出一个闭合的回路，使得每个城市刚好经过一次且仅一次且总的旅行距离最短。即要寻求一条回路 $T = (t_1, t_2, \ldots, t_n)$，使得下列目标函数最小：

$$f(T) = \sum_{i=1}^{n-1} d(t_i, t_{i+1}) + d(t_n, t_1)$$

  - 上式中 t_i为城市号，取值为 $[1, n]$，从而 $(t_1, t_2, \ldots, t_n)$ 就可以看作是关于n的一个排列。$d(t_i, t_j)$表示城市 t_i 与 t_j 之间的距离。对于对称型 TSP，有$d(t_i, t_j) = d(t_j, t_i)$.

- 旅行商问题的分类

- 从问题对应到图的类型，**TSP** 可以分为两类：

  1、任意两个城市间的距离都是对称的，它对应的是图论中的无向图；

  2、两个城市间的距离是非对称的，它对应的是图论中的有向图；

- 从问题本身的限制条件的强弱，主要有三类：

  1、不做任何限制(但是一般都要求城市间的费用不为负数)，只给出距离矩阵，求最短回路；

  2、要求距离间要满足三角不等式；

  3、定义在欧氏平面上的 **TSP**，即 **Euclidean TSP**，它给出每个城市在欧氏平面上的坐标，而城市间的距离就是以它们的欧氏距离来定义。

- 从问题的多项式可解性上分，**TSP** 可以分为两类：

  1、目前已经知道有多项式时间算法可解的，比如其距离矩阵满足特定的条件 **(Demidenko** 条件、**Kalmanson** 条件、**Supnick** 条件**)**等；

  2、目前尚没有发现多项式时间算法可解的，而研究热点是如何寻找更多的多项式时间可解的情形。

- 对旅行商问题的研究经过几十年的发展，已经产生了许多其它扩展形式,例如多旅行商问题**(Multi-Salesman Problem)**,多目标旅行商问题**(Multi-Objective TSP)**等等。

- 旅行商问题的应用和价值

- 旅行商问题是一个具有广泛的实用背景与重要的理论价值的组合优化难题。

- 许多关于 **TSP** 的工作并不仅是由实际应用直接推动的，而是因为 **TSP** 为其它一般的各类算法提供了思想方法平台，而这些算法广泛地应用于各种离散优化问题。

- 其次，**TSP** 大量的直接应用给研究领域带来了生机，并引导了未来的工作

- 运输问题是 **TSP** 最自然的应用。由于其模型的简单性，**TSP** 在其它一些领域有着有趣的应用。一个经典的例子是如何安排机器在一块电路板或其他物体上钻孔，其中需要钻的孔可以看成是各个城市，而旅行的费用就是钻头从一个孔移到下一个孔所花的时间。虽然钻孔的技术不断发展，但无论何时，只要钻机设备的移动时间在所有制造业的过程中占据显著的地位，**TSP** 在减少费用上就扮演了一个非常重要的角色。

- 许多实际中出现的问题都可以转化成旅行商问题的模型而解决。例如还有结晶学中的结构分析问题，车辆调度问题，计算机布线问题，单个机器上的工序调度问题等等。

- 旅行商问题的计算复杂性

- 时间复杂性，即随着输入问题规模的增长，算法所需计算步数的增长速度。

- 计算机科学家们有一个共识：即当输入规模**n**表示的算法复杂性函数 **f (n)**是以多项式为界的，算法才被认为是有效的。

- 从本质上讲，所有的计算问题又可以归结为判定问题，如果说：一个算法解决了某一判定问题，则算法输出“是”，否则输出“非”。而从输入到输出，算法所需要运行的步骤即为算法的时间复杂性。

- 很多优化问题，诸如旅行商问题、最小覆盖问题、多处理器任务调度问题、背包问题等都被发现可以多项式约化为 **NP** 中最难的问题，即 **NP** 完全问题。

- 普遍认为多项式时间的算法是“好的算法”、“有效的算法”，而所有的 **NP** 完全问题目前都还没有找到多项式时间的算法，它们需要耗费时间复杂性函数的数量级往往是指数级的，所以单单依靠提高计算机的速度对问题的解决是非常有限的

- **TSP的时间复杂度**

- **TSP 搜索空间随着城市数 n 的增大而增大，所有的旅程路线组合数为(n -1)!/2。**
- **5 个城市的情形对应 120/10=12 条路线；**
- **10 个城市的情景对应3628800/20=181440 条路线；**
- **100 个城市的情景对应有 4.666×10155 条路线。**
- **所以对于输入规模为n个城市的 TSP 找到最优解的时间复杂性函数的数量级是 O(n!)，当n 比较大时，耗费的时间已经是个天文数字。**

下表是在假定所用计算机每秒可以执行 **10** 亿次运算的前提下，对不同的时间复杂性函数所耗费时间的比较(**ns**表示纳秒， μs 表示微秒，**y**表示年，**c**表示世纪/百年）。

时间复杂度	输入量**n**				
函数	10	20	30	40	100
n^2	100ns	400ns	900ns	1.6μs	10μs
2^n	1.0μs	1.0ms	1.1s	18.3min	4.0c
$n!$	3.6μs	77.1y	$8.4\times 10^{13}c$	$2.6\times 10^{29}c$	$3.0\times 10^{139}c$

- 求解旅行商问题的已有算法

- 多年来对 **TSP** 的研究，人们提出了许多求解方法，其中有精确算法如线性规划方法、动态规划方法、分支定界方法；

- 近似算法如插入法、最近邻算法、**Clark&Wright** 算法、生成树法、**Christofides** 算法、**r-opt** 算法、混合算法、概率算法等。

- 近年来，还有很多尝试解决该问题的较为有效的方法不断被提出，例如禁忌搜索方法、遗传算法、模拟退火算法、**Hopfield**神经网络方法、蚁群算法、免疫算法等。

7.2 旅行商问题的数学规划模型

设城市的个数为n，d_{ij}是两个城市i与j之间的距离，$x_{ij}=0$或 1（1 表示走过城市i到城市j的路，0 表示没有选择走这条路）。则有

$$\min \sum_{i\neq j} d_{ij}x_{ij},$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij}=1,\ i=1,2,\cdots,n,\ (\text{每个点只有一条边出去}),$$

$$\sum_{i=1}^{n} x_{ij}=1,\ j=1,2,\cdots,n,\ (\text{每个点只有一条边进去}),$$

$$\sum_{i,j\in s} x_{ij}\leq |s|-1,\ 2\leq |s|\leq n-1,\ s\subset\{1,2,\cdots,n\},\ \text{即}s\text{为}\{1,2,\cdots,n\}\text{的真子集},$$

（除起点和终点外，各边不构成圈）

$$x_{ij}\in\{0,1\},\ i,j=1,2,\cdots,n,\ i\neq j.$$

- 应用举例

例1　设备更新问题。某企业使用一台设备，在每年年初都要决定是购置新设备还是继续使用旧的。购置新设备要支付一定的购置费，使用旧设备则要支付维修费。制定一个五年内的设备更新计划，使得总支付费用最少。

已知该设备在各年年初的价格为：

第一年	第二年	第三年	第四年	第五年
11	**11**	**12**	**12**	**13**

已知使用不同时间的设备维修费用为：

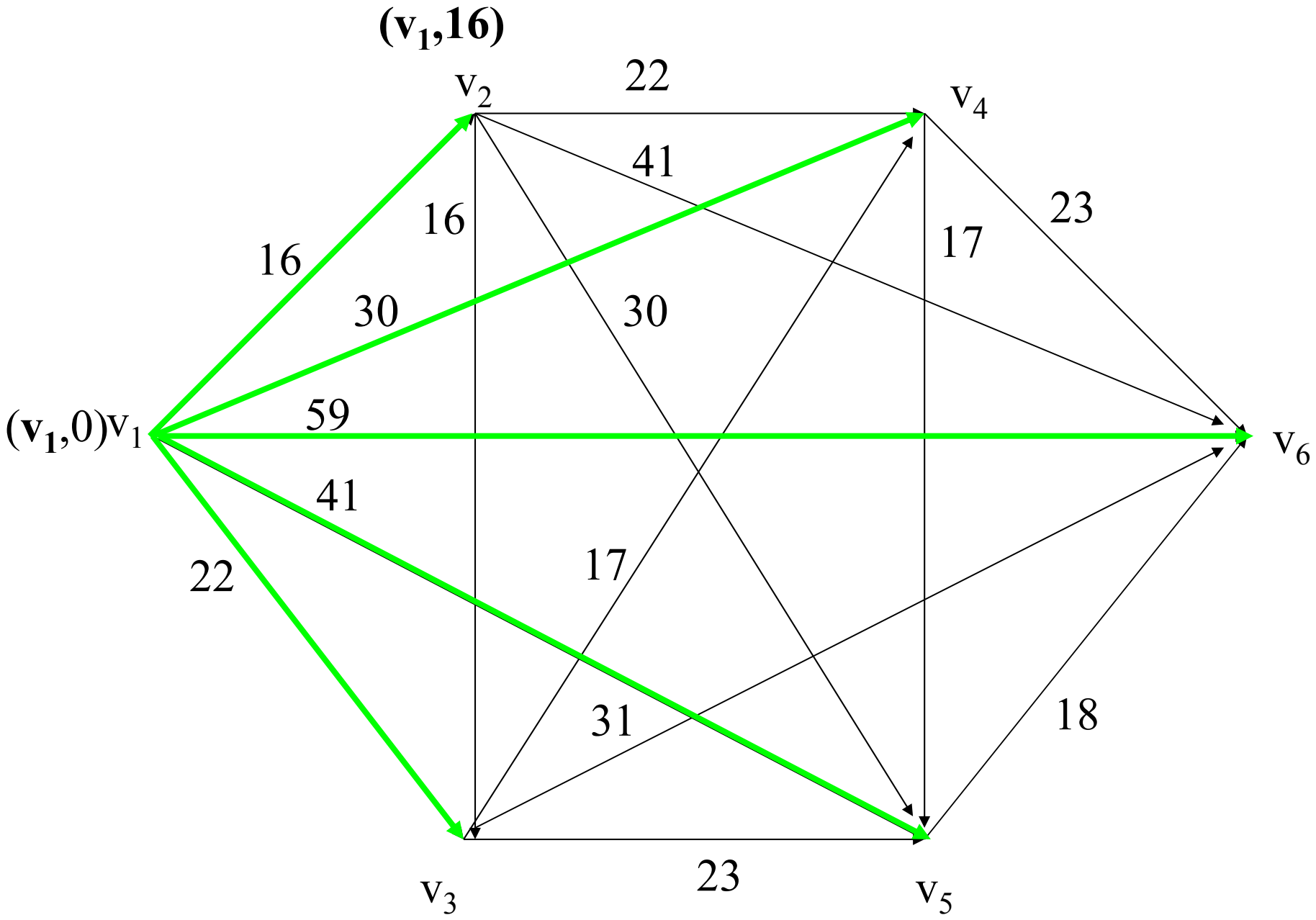
使用年数	**0~1**	**1~2**	**2~3**	**3~4**	**4~5**
维修费用	**5**	**6**	**8**	**11**	**18**

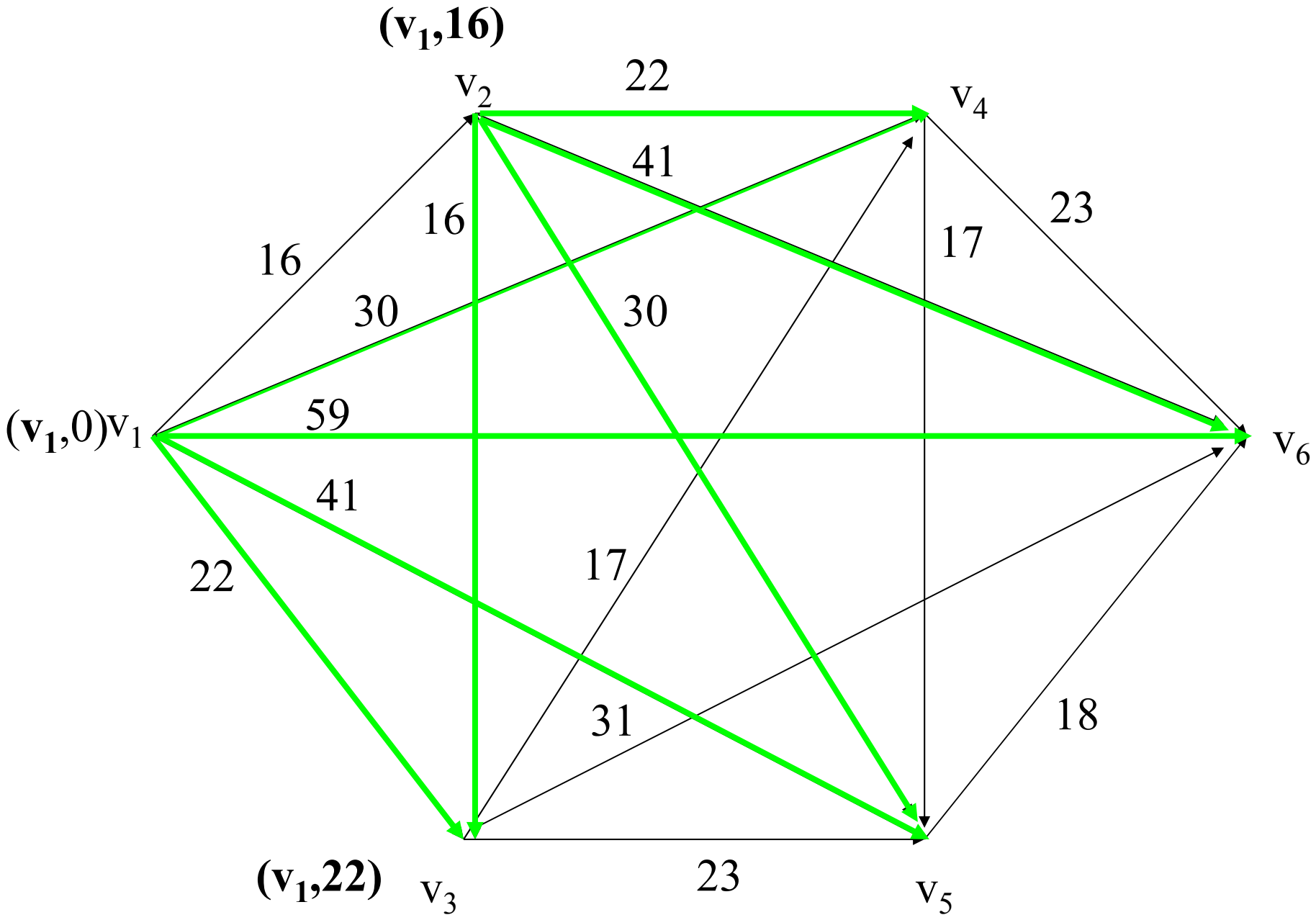
设以$v_i(i=1,2,3,4,5)$表示“第i年初购进一台新设备”这种状态，以v_6表示“第5年末”这种状态；以弧(v_i, v_j)表示“第i年初购置的一台设备一直使用到第j年初”这一方案，以w_{ij}表示这一方案所需购置费和维修费之和。

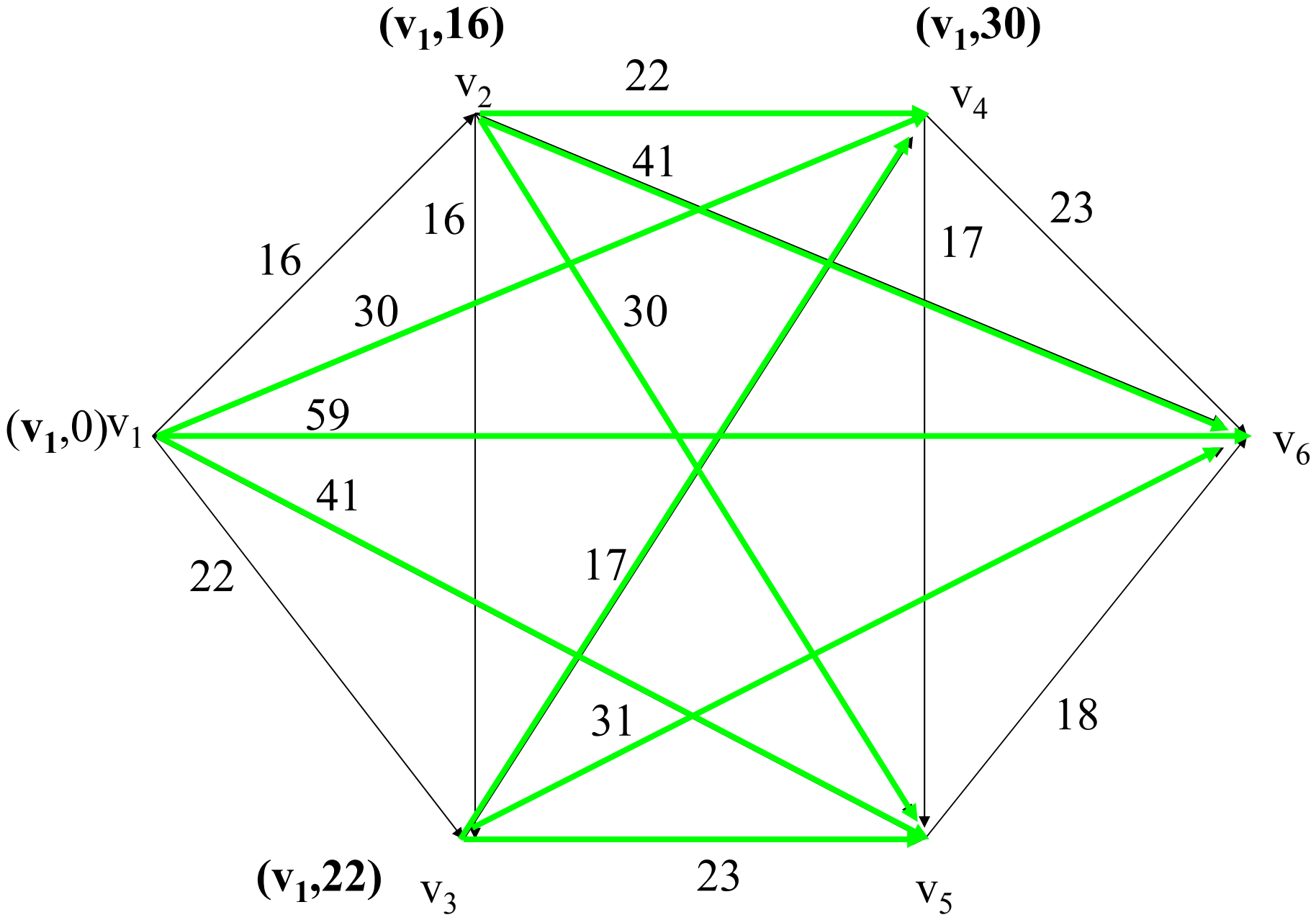
这样，可建立本例的网络模型。于是，该问题就可归结为从图中找出一条从v_1到v_6的最短路问题。

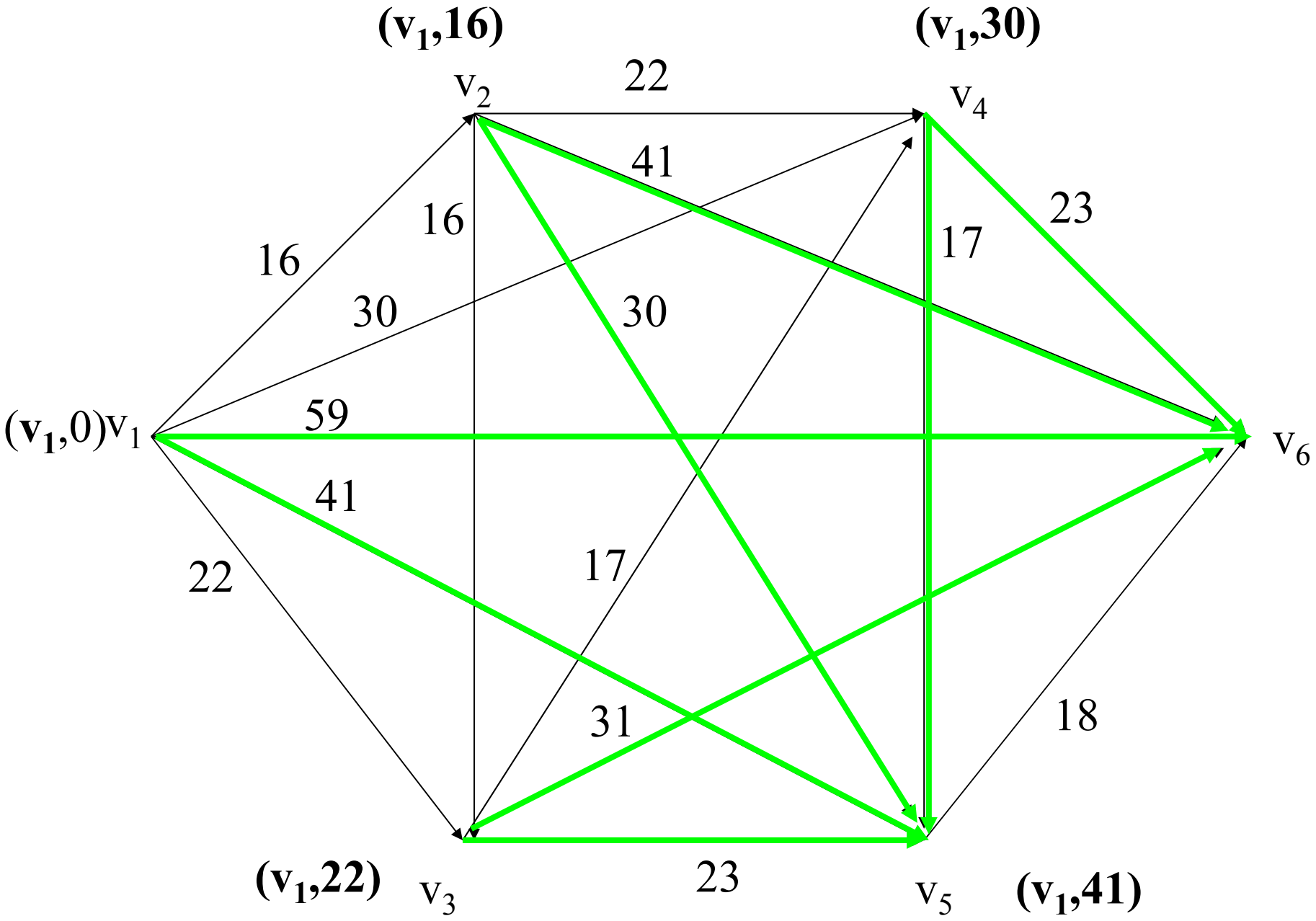
从图中可以得出两条最短路：

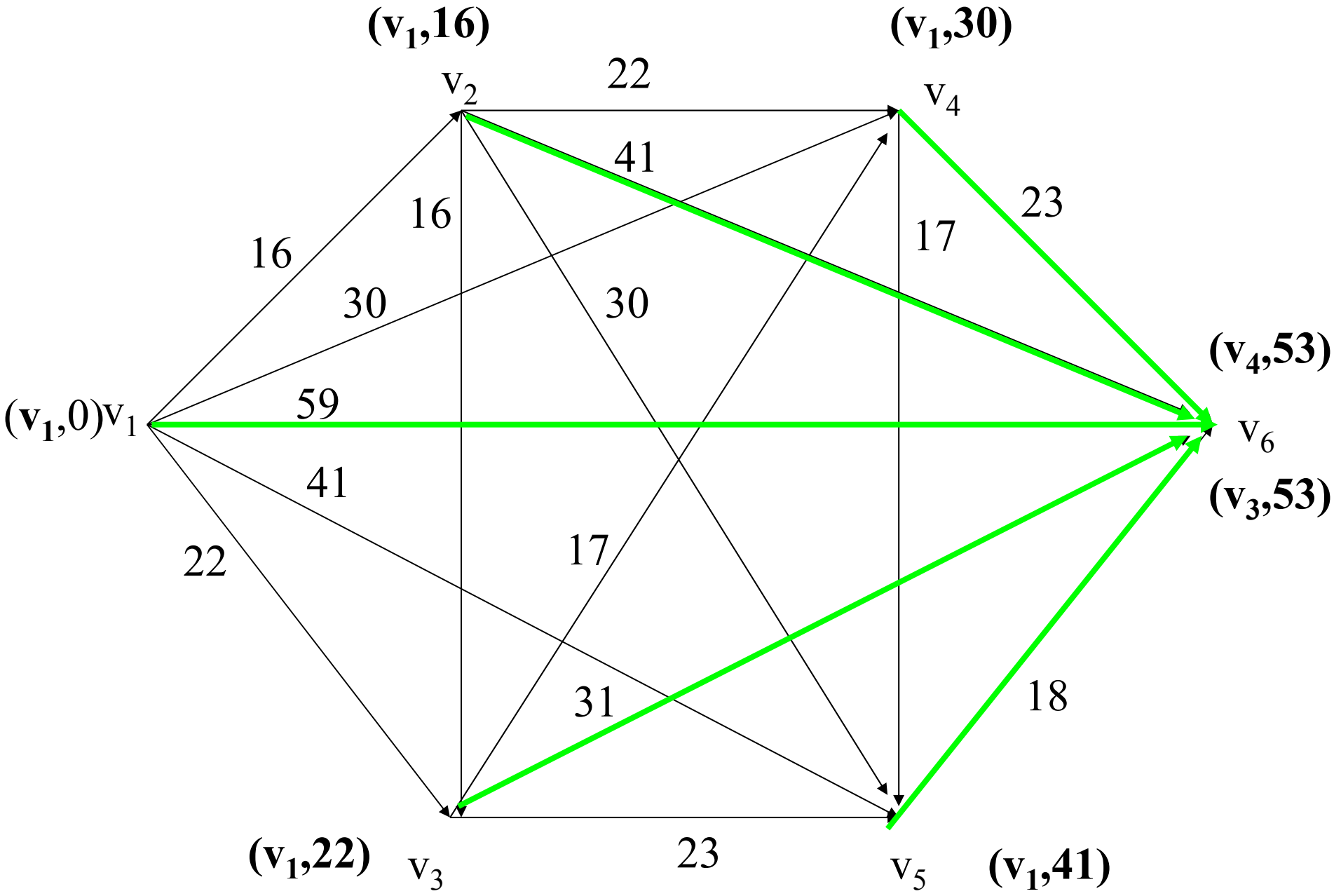
v_1 —v_3—v_6　　；　v_1 —v_4—v_6


(v1,16)
v2
22
v4
41
16
16
23
17
30
30
(v1,0)v1
59
v6
41
22
17
31
18
v3
23
v5


(v1,16)
v2
22
v4
41
16
16
23
17
30
30
(v1,0)v1
59
v6
41
22
17
31
18
(v1,22)
v3
23
v5


(v1,16)
(v1,30)
v2
22
v4
41
16
16
17
23
16
30
30
(v1,0)v1
59
v6
41
22
17
31
18
(v1,22)
v3
23
v5


(v1,16)
(v1,30)
v2
22
v4
41
16
16
17
23
16
30
30
(v1,0)v1
59
v6
41
22
17
31
18
(v1,22)
v3
23
v5
(v1,41)


(v1,16)
(v1,30)
v2
22
v4
41
16
16
23
17
16
30
30
(v4,53)
(v1,0)v1
59
v6
41
(v3,53)
22
17
31
18
(v1,22)
v3
23
v5
(v1,41)