运筹学

13. 网络分析

李 力
清华大学

Email: li-li@tsinghua.edu.cn

2023.12.

主要内容

基本概念

最小支撑树

最短路

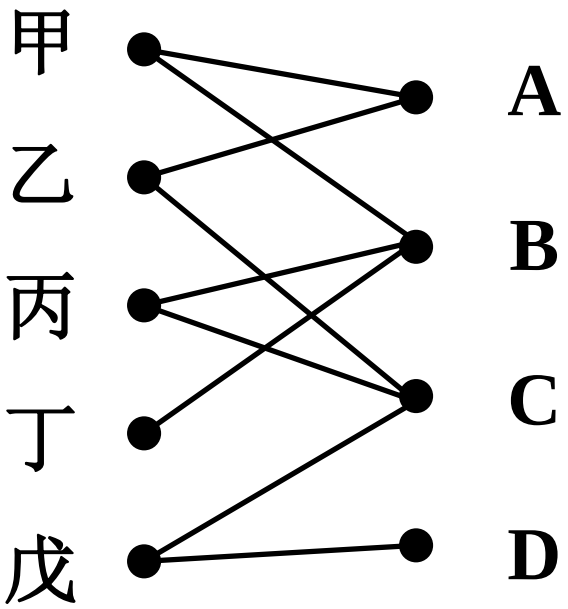
基本概念

图的定义

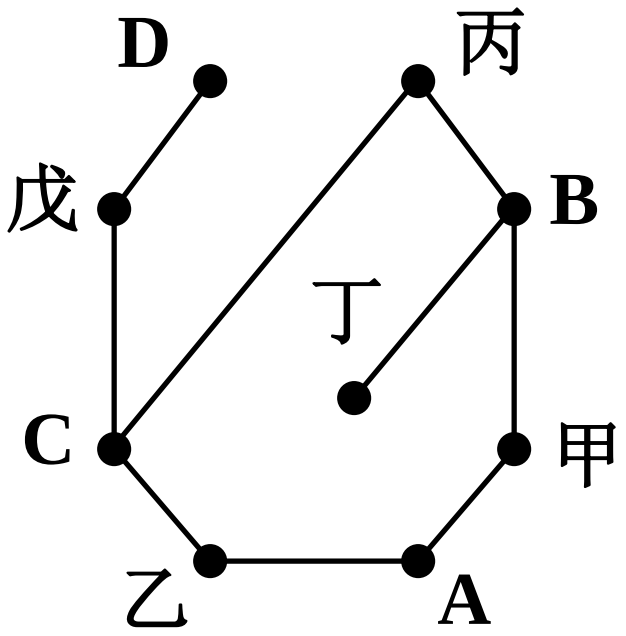
点集 V $\rightarrow$ 边集 E $\rightarrow$ 图 $G=(V,E)$

例 $V=\{$甲，乙，丙，丁，戊，A，B，C，D$\}$

$$E=\left\{\begin{array}{l}(甲,A),(甲,B),(乙,A),(乙,C),(丙,B)\\(丙,C),(丁,B),(戊,C),(戊,D)\end{array}\right\}$$



$\Leftrightarrow (V,E) \Leftrightarrow$

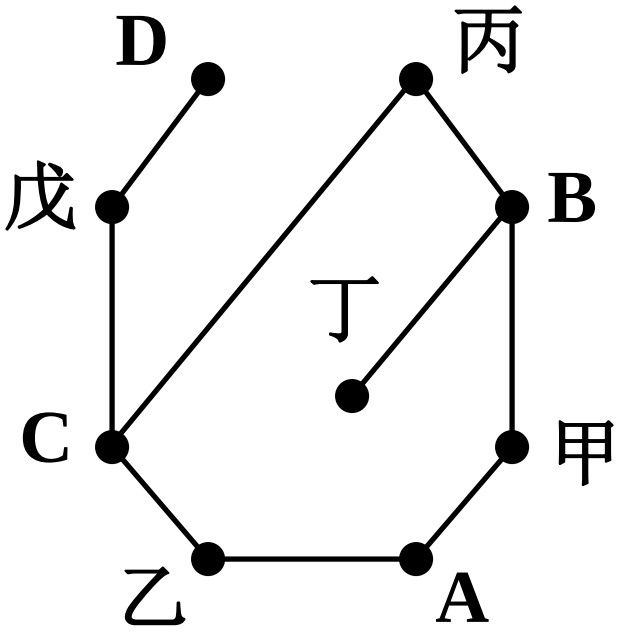


根据维基百科：Network theory is a part of graph theory: a network can be defined as a graph in which nodes and/or edges have attributes (e.g. names).

图（graph）是高度抽象后的用邻接矩阵（adjacency matrix）所包含的结构信息，而网络（network）在此之上可以通过在边（edge）和点（vertice）上增加属性来包含更多的信息，更偏重于对实际问题的建模。

无向图

$(甲, A) = (A, 甲)$

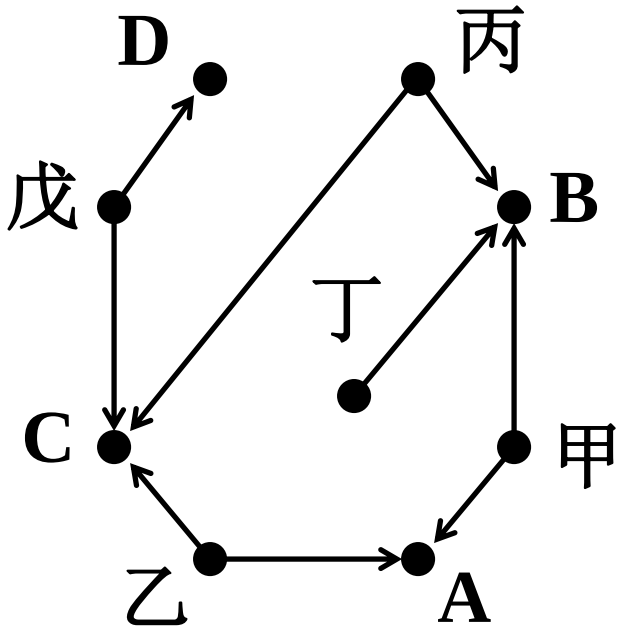


有向图

$(甲, A) \neq (A, 甲)$

甲 → A → 甲

始点　　终点



教材将有向图的边称为弧（arc），以示区别

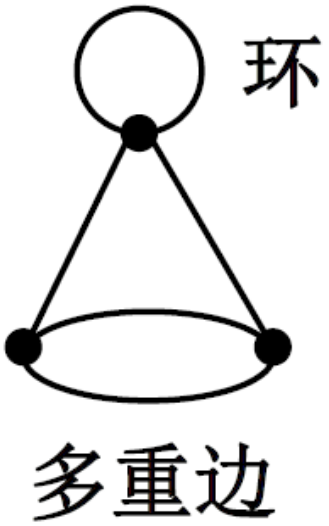
相邻 如果 $v_1, v_2 \in V$, $(v_1, v_2) \in E$，称 v_1, v_2 相邻，称 v_1, v_2 为 (v_1, v_2) 的端点

如果 $e_1, e_2 \in E$，并且有公共端点 $v \in V$，称 e_1, e_2 相邻，称 e_1, e_2 为 v 的关联边

对 $G = (V, E)$，$m(G) = |E|, n(G) = |V|$ 表示边数和点数

自回路 两端点相同的边，或称为环

多重边 两点之间多于一条不同的边

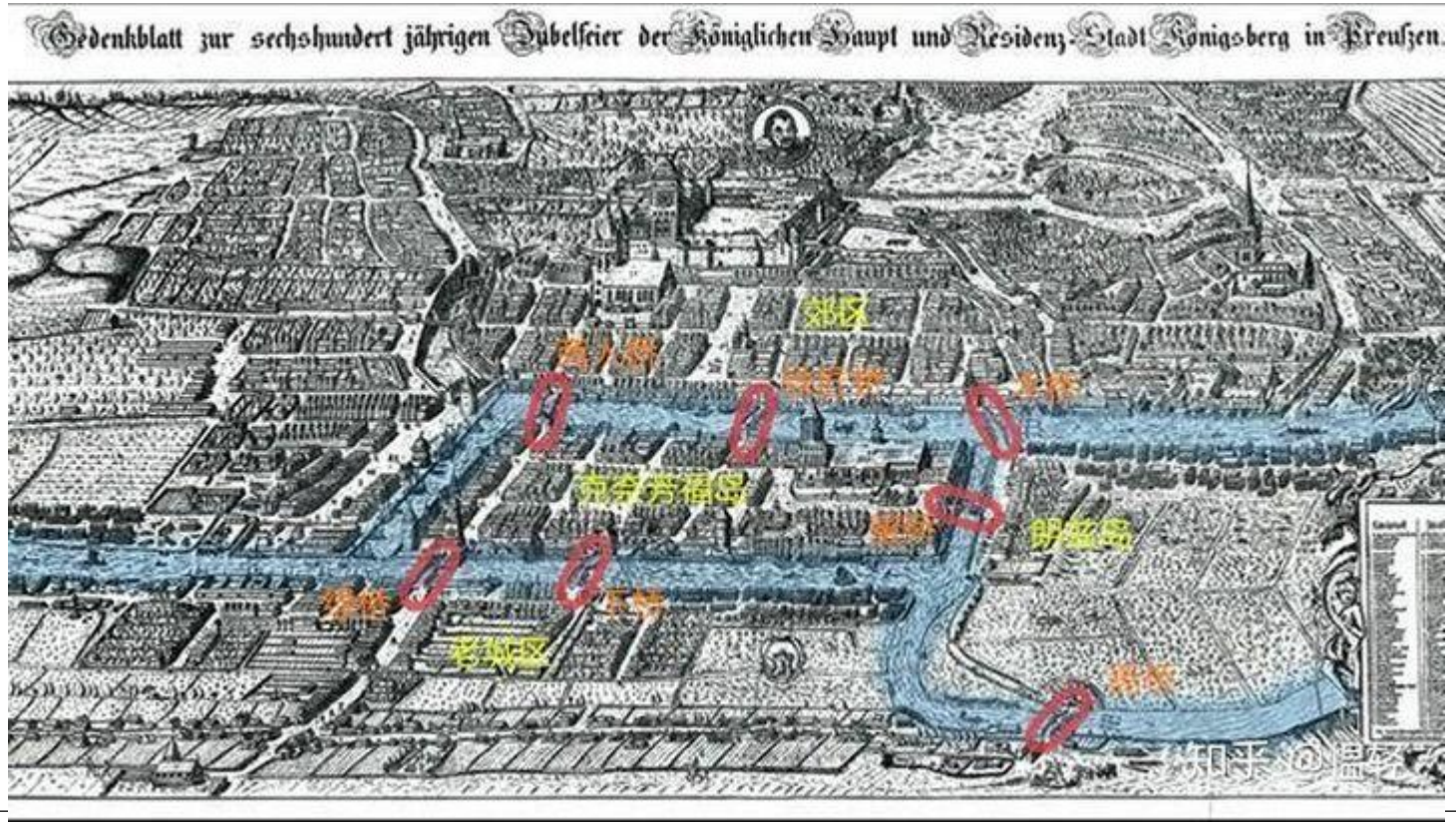


简单图与多重图

简单图 不含自回路和多重边的图

多重图 含有多重边的图

1731年，年轻的欧拉正在俄国的彼得堡科学院担任物理教授，他接收到一封看似随意的询问信，里面包含着一个在当时广为人知但却却令许多人困惑的问题：在普鲁士的哥尼斯堡有一个被称为奈发夫的岛屿；普雷格尔河的两条支流从岛的两旁流过，且有七座桥横跨这两条支流。当时哥尼斯堡的居民中流传着这样一道难题：一个人如何才能一次走遍七座桥，且每座桥只走过一次，最后回到出发点？



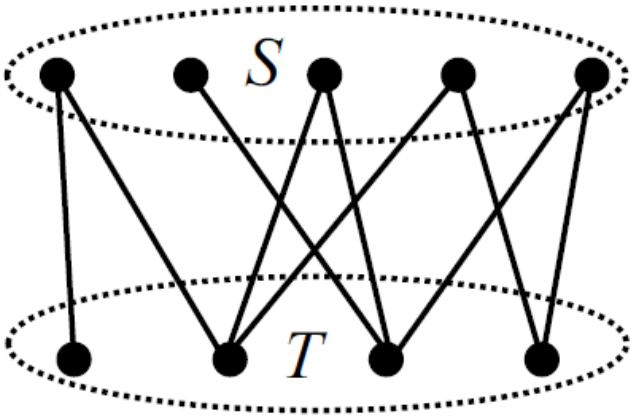
1735年8月26日，欧拉向当时俄国的圣彼得堡科学院递交了一篇名为《有关位置几何的一个问题的解》的论文，阐述了他是如何否定哥尼斯堡七桥问题能一次走完的。

在文中，他还就此类问题进行了一般性意义的讨论：“若连接奇数座桥的地点多于两个，则找不到符合要求的路线；若仅有两个地点与奇数座桥相连，则可从这两个地点的任意一个出发，找出符合要求的路线；若无一地点是通往奇数座桥的，则无论从哪个地点出发，都能找到所要求的路线。

这也是首次人们开始认识到可以用点与线来描述具体问题，直接引发了后续对图论的继续研究。

完全图 任意两个顶点间都有边相连的无向简单图称为完全图，有 n 个顶点的无向完全图记为 K_n ，任意两个顶点间都有且仅有一条边相连的有向简单图称为有向完全图

二分图 如果 V 可以划分为两个不相交的子集 $S,T,$ 使得 E 中每条边的两个端点必有一个属于 S ，一个属于 T ，称 $G=(V,E)$ 为二分图记为 $G=(S,T,E)$ 如右图



端点的次 以点 v 为端点的边数称为 v 的次，记为 $\deg(v)$ ，或简记为 $d(v)$

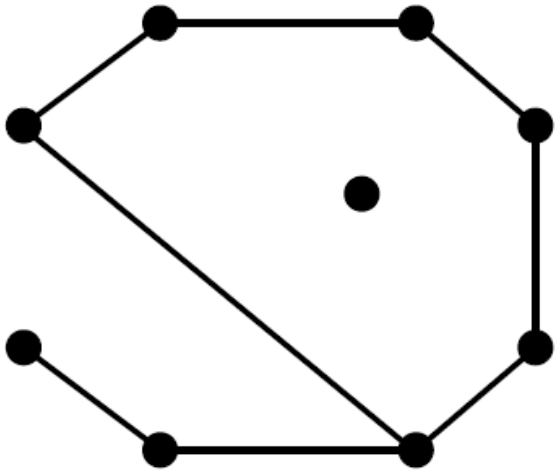
次为 0 的点称为孤立点，次为 1 的点称为悬挂点，连接悬挂点的边称为悬挂边，次为奇数的点称为奇点，次为偶数的点称为偶点

出次与入次

在有向图中，以 v 为始点的边数称为 v 的出次，用 $d^+(v)$ 表示，以 v 为终点的边数称为 v 的入次，用 $d^-(v)$表示

定理1 任何图中，顶点次数总和等于边数的 2 倍

定理2 任何图中，奇点的个数为偶数个



顶点次数总和等于 16

边数等于 8

奇点的个数为 2 个

定理1显然成立，下面证明定理2：根据定理1

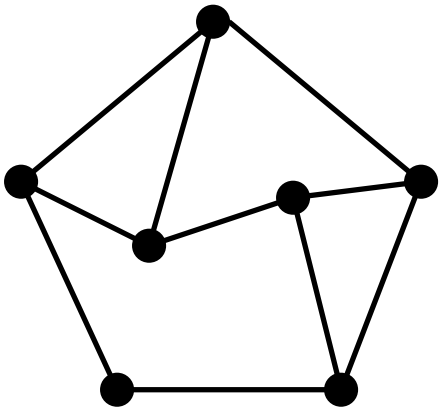
奇点次数总和 ＋ 偶点次数总和 ＝ 偶数

（偶数）

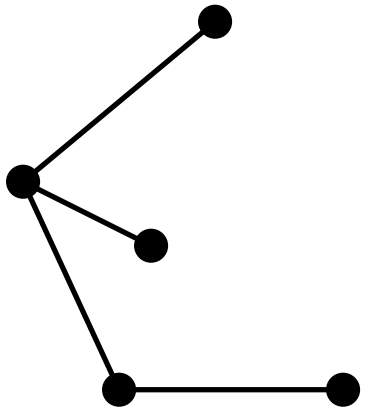
若有奇数个奇点，其次数总和为奇数，上式不成立

子图

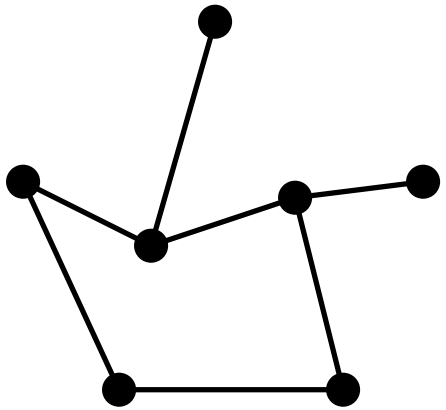
对于图 $G=(V,E)$，如果 E' 是 E 的子集，V' 是 V 的子集，并且 E' 中每条边的端点都属于 V'，则称 $G'=(V',E')$ 是 G 的子图，如果 $V'=V$，则称 G' 是 G 的支撑子图



图



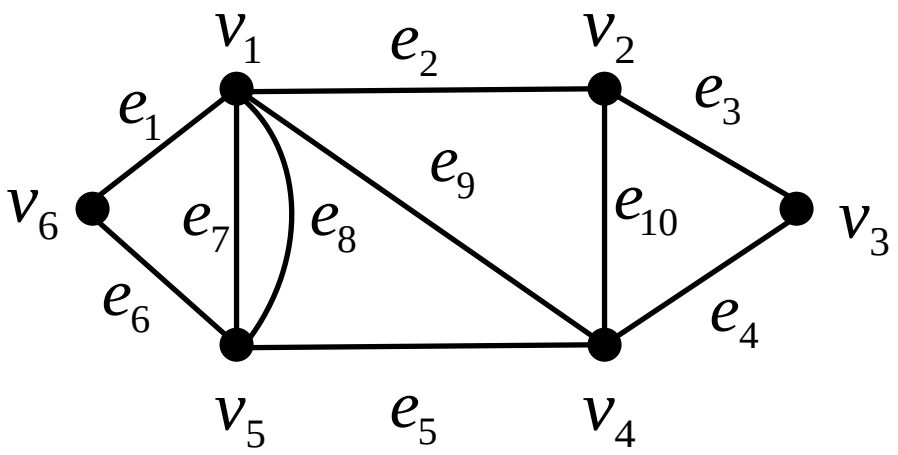
子图



支撑子图

路（链） 点、边交替（可重复）的序列，如

$$S=\{v_6,e_6,v_5,e_7,v_1,e_8,v_5,e_7,v_1,e_9,v_4,e_4,v_3\}$$



无重复边的路称为简单路，无重复点的路为初级路

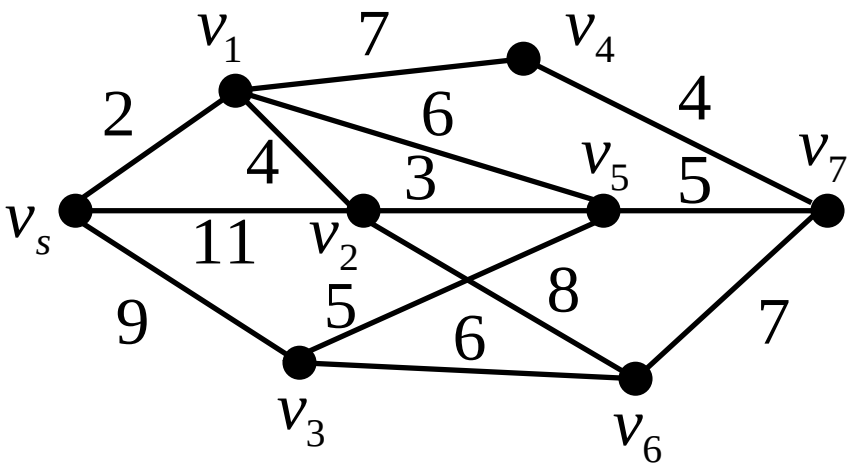
始点和终点为同一点的路称为回路

图的两点间若存在连接两点的一条路，称这两点连通

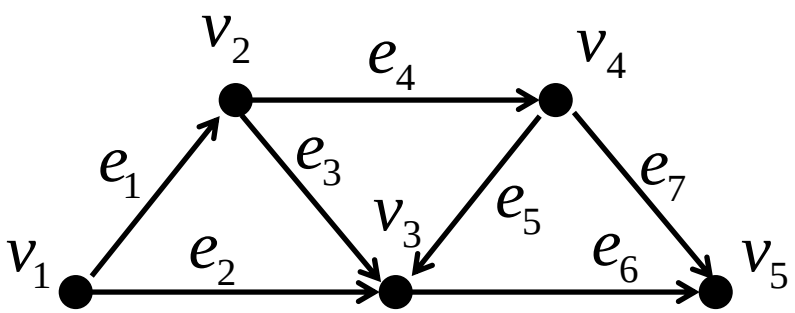
任意两点都连通的图称为连通图

网络（赋权图）

每边赋有权（实数或实数向量）的图称为网络，记为 $G=(V,E,W)$，其中 $W=\{w(e),\ e\in E\}$ 是权的集合，无向图赋权构成无向网络，有向图赋权构成有向网络，下图为一个无向网络



有向图的关联矩阵



$$
\begin{array}{c}
 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\
\begin{pmatrix}
1 & 1 & & & & & \\
-1 & & 1 & 1 & & & \\
 & -1 & -1 & & -1 & 1 & \\
 & & & -1 & 1 & & 1 \\
 & & & & & -1 & -1
\end{pmatrix}
\end{array}
$$

将有向图关联矩阵的-1换成1就得到无向图关联矩阵

图的邻接矩阵

两点间有边为1，否则为0

$$
\begin{array}{c}
 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0
\end{pmatrix}
\end{array}
$$

求解如下线性规划问题：

$$\min C^T X$$

$$s.t.\ \ AX = B,\ 0 \le X \le D$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 10 \\ 0 \\ 0 \\ 0 \\ -10 \end{bmatrix}$$

$$C^T = \begin{bmatrix} 4 & 1 & 2 & 6 & 3 & 1 & 2 \end{bmatrix}$$

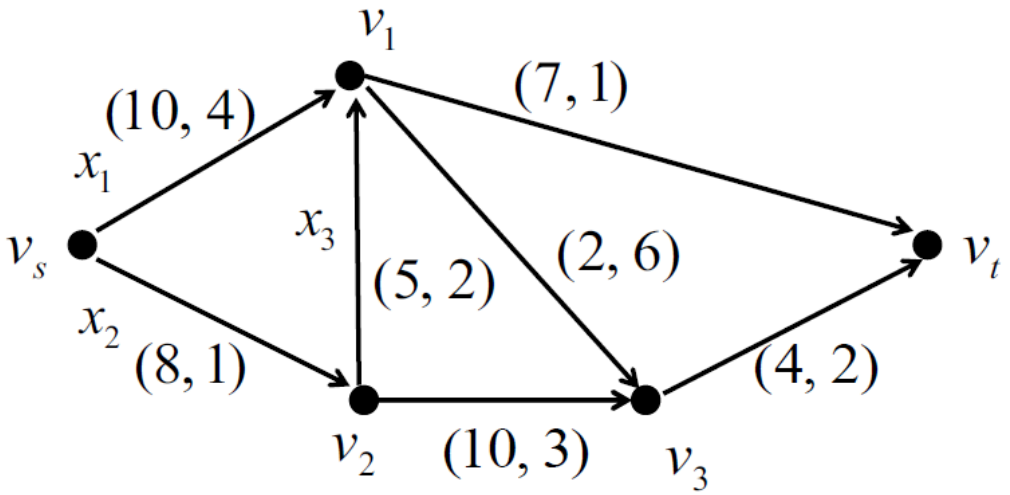
$$D^T = \begin{bmatrix} 10 & 8 & 5 & 2 & 10 & 7 & 4 \end{bmatrix}$$

求解如下线性规划问题：

$$\min C^T X$$

$$s.t.\ \ AX = B,\ 0 \le X \le D$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$



$$B^T = [10 \quad 0 \quad 0 \quad 0 \quad -10]$$

$$C^T = [4 \quad 1 \quad 2 \quad 6 \quad 3 \quad 1 \quad 2]$$

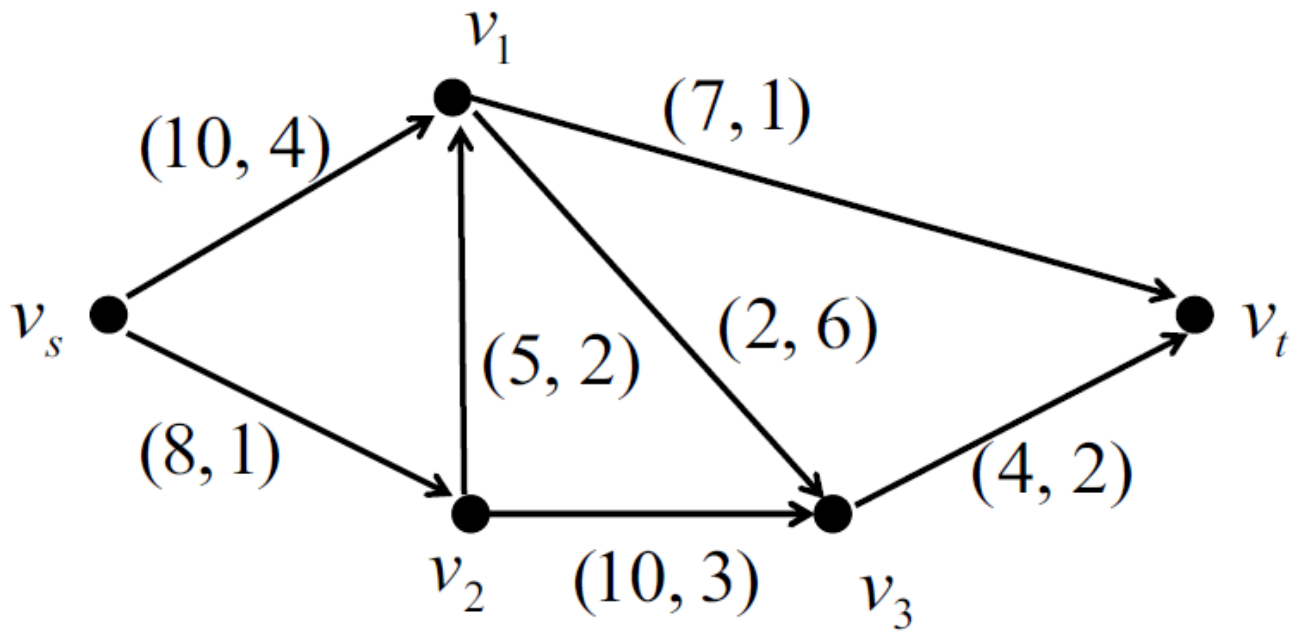
$$D^T = [10 \quad 8 \quad 5 \quad 2 \quad 10 \quad 7 \quad 4]$$

$$v_s:\ \ x_1 + x_2 = 10$$

$$v_1:\ \ -x_1 - x_3 + x_4 + x_6 = 0$$

$$\vdots$$

等价于如下最小费用流问题



括号内第一个数字是容量，第二个是单位流量费用

目标：从发点到收点的总的流量费用最小

约束：1）容量约束，各边流量不大于容量

2）流量平衡约束，各点进出流量总和相等

3）从发点到收点的总流量为10

如下例

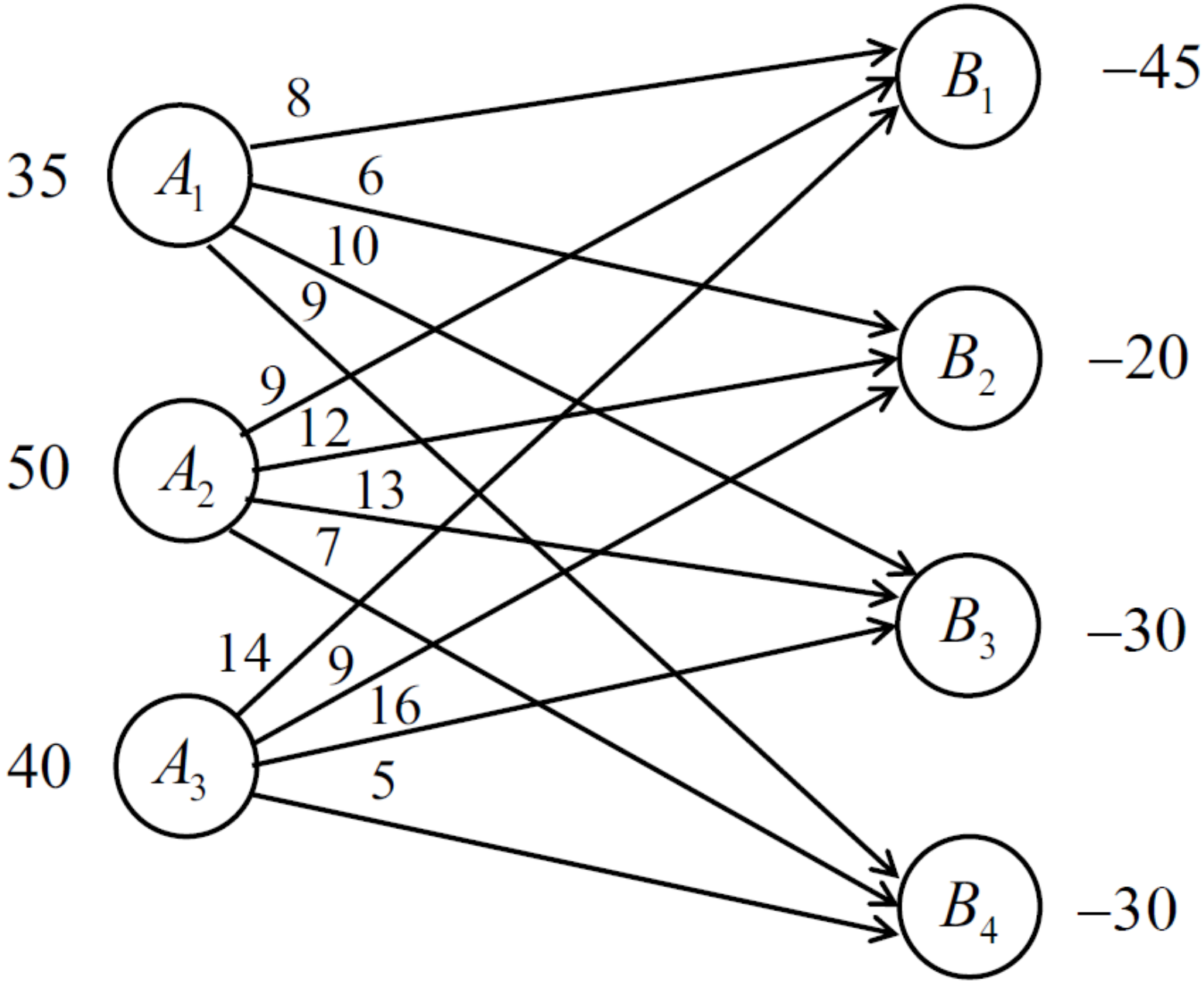
销地 / 产地	B_1	B_2	B_3	B_4	产量
A_1	8 x_{11}	6 x_{12}	10 x_{13}	9 x_{14}	35
A_2	9 x_{21}	12 x_{22}	13 x_{23}	7 x_{24}	50
A_3	14 x_{31}	9 x_{32}	16 x_{33}	5 x_{34}	40
销量	45	20	30	30	

数学规划模型为

$$\min \sum_{i=1}^{3}\sum_{j=1}^{4} c_{ij}x_{ij} = 8x_{11}+6x_{12}+10x_{13}+9x_{14}+9x_{21}+12x_{22}+13x_{23}$$
$$+7x_{24}+14x_{31}+9x_{32}+16x_{33}+5x_{34}$$

$$\text{s.t.}\quad x_{11}+x_{12}+x_{13}+x_{14}=35$$
$$x_{21}+x_{22}+x_{23}+x_{24}=50$$
$$x_{31}+x_{32}+x_{33}+x_{34}=40$$
$$x_{11}+x_{21}+x_{31}=45$$
$$x_{12}+x_{22}+x_{32}=20$$
$$x_{13}+x_{23}+x_{33}=30$$
$$x_{14}+x_{24}+x_{34}=30 \qquad x_{ij}\geq 0,\ \forall 1\leq i\leq 3, 1\leq j\leq 4$$

怎样调运这些物品能使总费用最小？



树

树

设 $T=(V,E)$，顶点个数 $|V|=n$，如果下述任何一条满足，T 就是一个树（定理6.3.1，195页）

王焕钢老师
ppt要求n>=3

1）T 连通且无回路

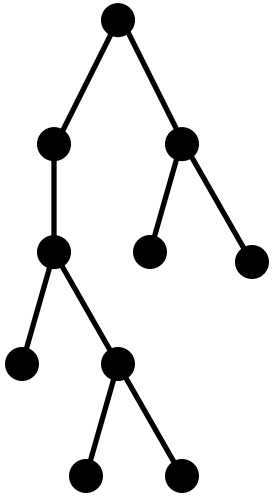
2）T 无回路且有 $n-1$ 条边

3）T 连通且有 $n-1$ 条边

4）T 连通且每条边都是割边

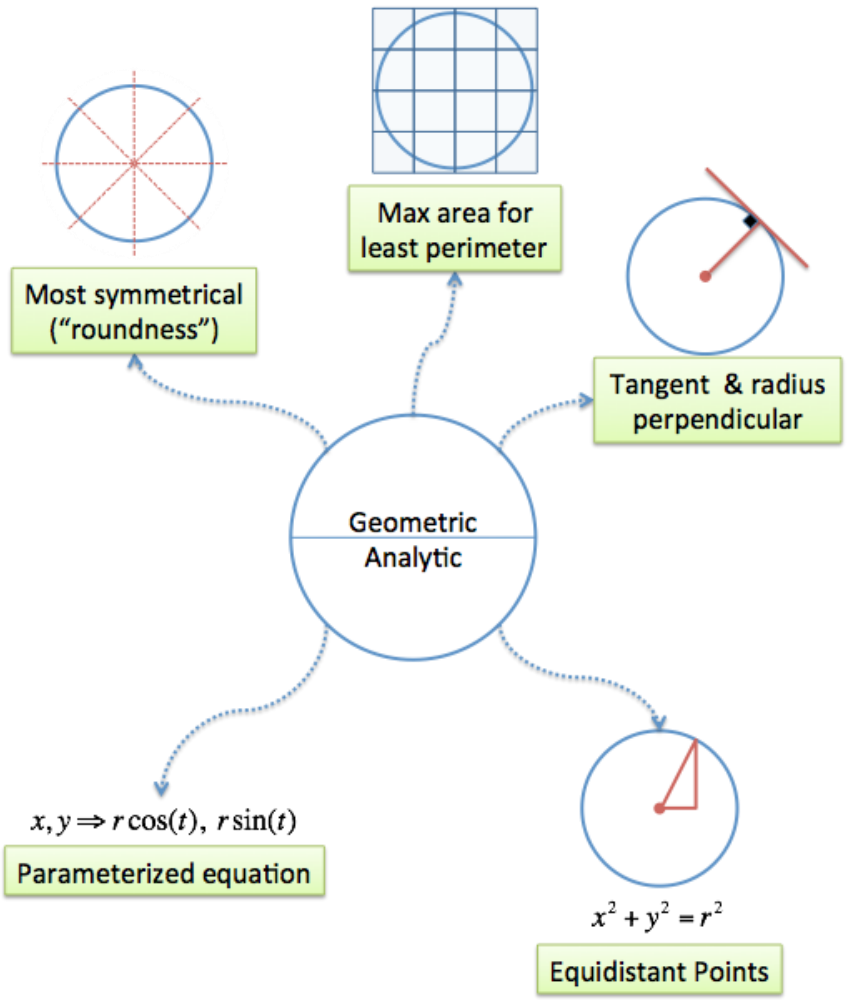
5）T 的任意两点都有唯一的路连接

6）T 无回路，但任意加上一个新边就构成唯一回路



$n=10$

Defining a Circle



很显然圆的定义有无数种，列出几种非常典型的：

- 所有的二维形状中，最对称的图形；
- 周长一定的情况下，面积最大的图形；
- 图形上的点距离一点（圆心）长度相同；
- 满足公式 $x^2 + y^2 = r^2$ ；
- 图形上的点可以定义为： $r \cdot cost$ ， $r \cdot sint$
- 图像上任意点的切线都是垂直于这一点的位置矢量

https://betterexplained.com/articles/developing-your-intuition-for-math/

定理 2 每个树至少有两个次为 1 的点。

若 $T=(V,E)$ 恰好有两个次为 1 的点，则其它点的次必为 2，因此 $T=(V,E)$ 是一条链（路）

假设只有0个或者1个次为1的点（叶子），那么剩下的点的次都>=2，所以图中点的总次数>= 2(n-1) + 1。但树只有n-1条边，这就矛盾了

类似的，根据只有n-1条边这个性质，可以推出，如果恰好有两个次为1的点，则其它点的次必为2，又根据连通性，得出是一条链

图的支撑树 如果 $G=(V,E)$ 的支撑子图 $T=(V,E')$ 是树，称其为 G 的支撑树，G 中属于 T 的边称为树枝，不属于 T 的边称为弦

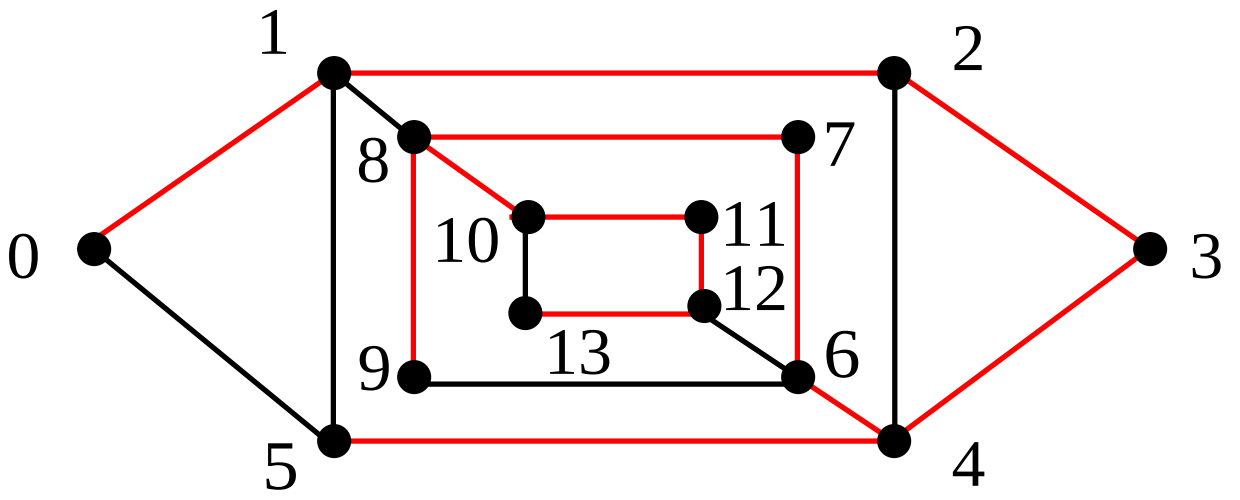
命题： 图 $G=(V,E)$ 有支撑树的充要条件是 G 是连通图

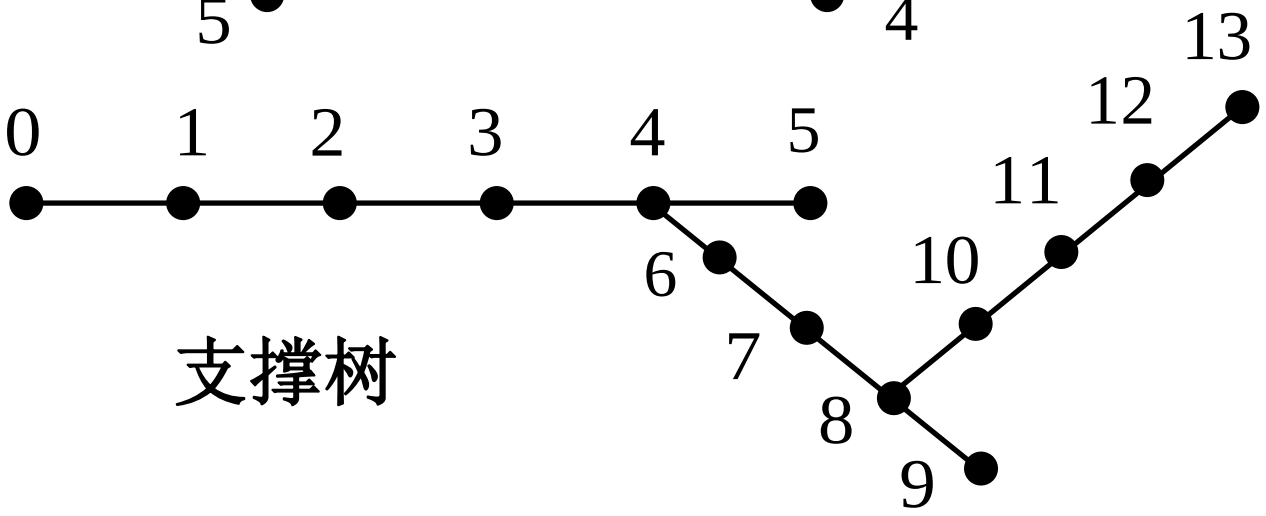
必要性是显然的，充分性的证明等价于对任意的连通图给出确定支撑树的算法

确定支撑树的方法

1）深探法

从任意点开始，边标号边前进，只至标完所有点

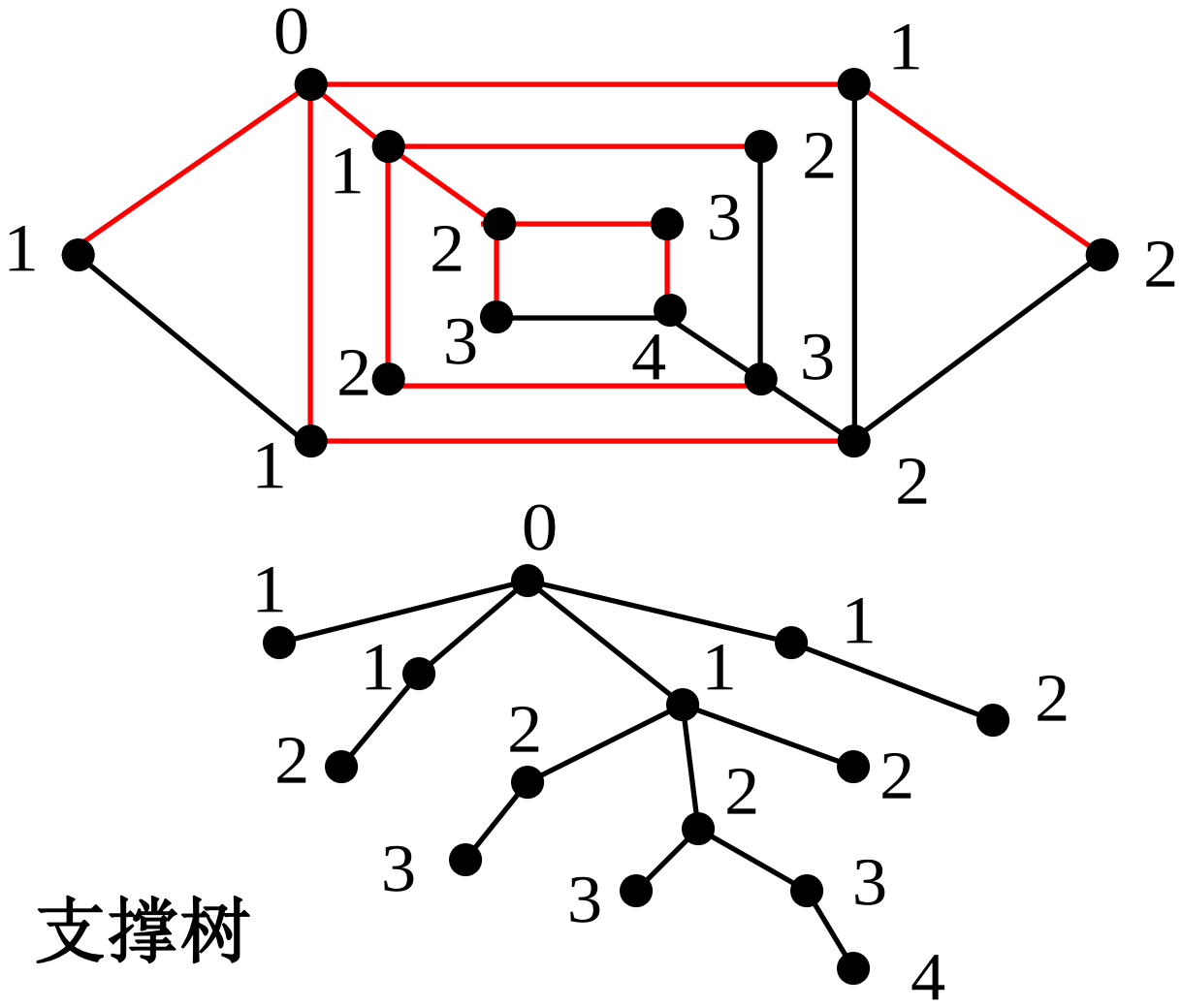




支撑树

2）广探法

从任意点开始，把当前标号点附近标号完再前进



支撑树

最小支撑树

最小支撑树

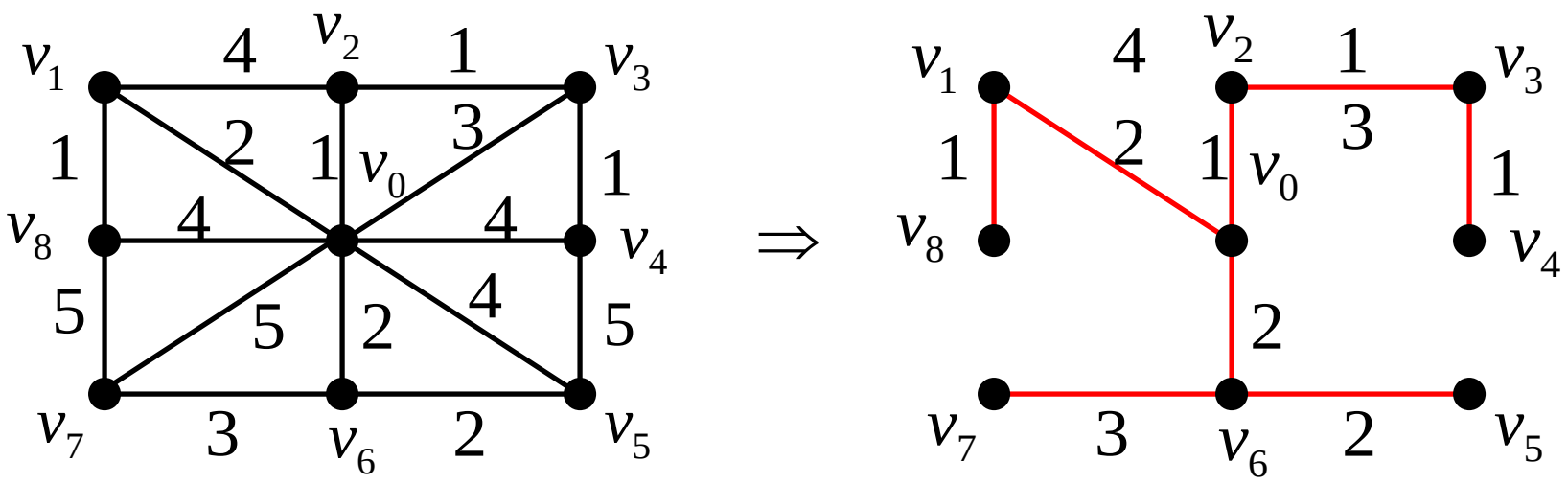
网络 $G=(V,E,W)$ 的任意一个支撑树 T 的所有树枝上的权的总和，记为 $L(T)$，称为这个支撑树的权，具有最小权的支撑树称为最小支撑树，简称最小树

最小支撑树在交通网、电力网等设计中广泛应用

求最小支撑树的Kruskal算法（避圈法）

将所有边按权值从小到大排序，从权值最小的边开始选树枝，如果可能形成回路（圈）则跳过，直至选够顶点数减1的树枝

例



所有边从小到大排列

$$(v_0,v_2)=1,(v_2,v_3)=1,(v_3,v_4)=1,(v_1,v_8)=1,(v_0,v_1)=2$$
$$(v_0,v_6)=2,(v_5,v_6)=2,(v_0,v_3)=3,(v_6,v_7)=3,(v_0,v_4)=4$$
$$(v_0,v_5)=4,(v_0,v_8)=4,(v_1,v_2)=4$$

从小到大顺序选择不构成回路的边形成右上支撑树

性质：加入任何弦形成的回路中，弦的权值最大

定理：*T* 是最小支撑树的充要条件是：加入任何弦形成的回路中，弦的权值最大

$\Rightarrow$ **Kruskal算法产生的是最小支撑树**

证明必要性：

如果加入某个弦形成的回路中有比该弦的权值更大的树枝，则用该弦代替最大树枝形成的支撑树的总权值会变小，和最小支撑树定义矛盾

证明充分性：

设 T_1 是满足条件的支撑树，T_2 是所有最小支撑树中和 T_1 不同的树枝数最少的树（一定存在），记

$$T_1 = \left\{e_1, e_2, \cdots, e_m, \hat{T}\right\}_2, \quad T_2 = \left\{\overline{e}_1, \overline{e}_2, \cdots, \overline{e}_m, \hat{T}\right\}$$

其中 $w(\overline{e}_1) \le w(\overline{e}_i),\ \forall 2 \le i \le m$

将 $\overline{e}_1$ 加入 T_1 会形成回路，一定有 $e_k \in T_1 \setminus \hat{T} \Rightarrow w(e_k) \le w(\overline{e}_1)$

将 e_k 加入 T_2 会形成回路，一定有 $\overline{e}_j \in T_2 \setminus \hat{T}$

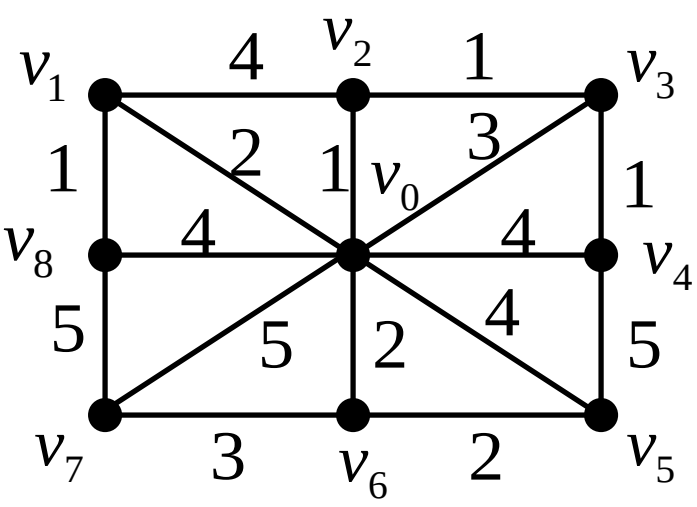
$$w(e_k) \le w(\overline{e}_1) \le w(\overline{e}_j) \quad \Rightarrow \quad T_2(e_k \setminus \overline{e}_j) \text{ 仍是最小支撑树}$$

$T_2(e_k \setminus \overline{e}_j)$ 和 T_1 的不同树枝数为 $m-1$，矛盾！ $\Rightarrow \quad m = 0$

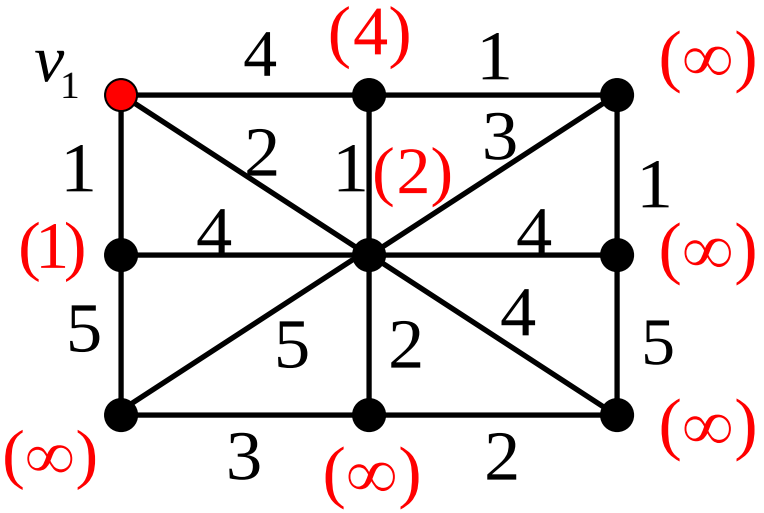
求最小支撑树的Dijkstra算法

从任意点开始逐渐增加某个点集，记为 S，每次从不在 S 的点集里选择距 S 一步距离最小的点加入 S，将相应边取为树枝，直至 S 包含所有的点

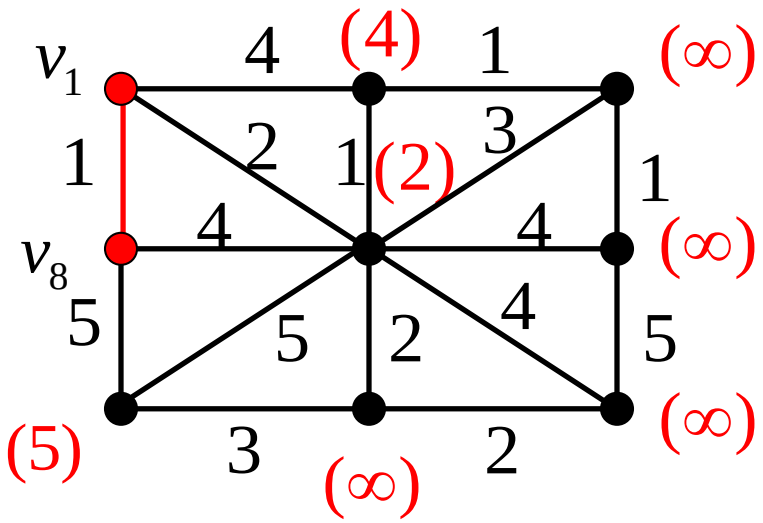
例 （下面的红点构成 S 集，红数是距 S 的一步距离）



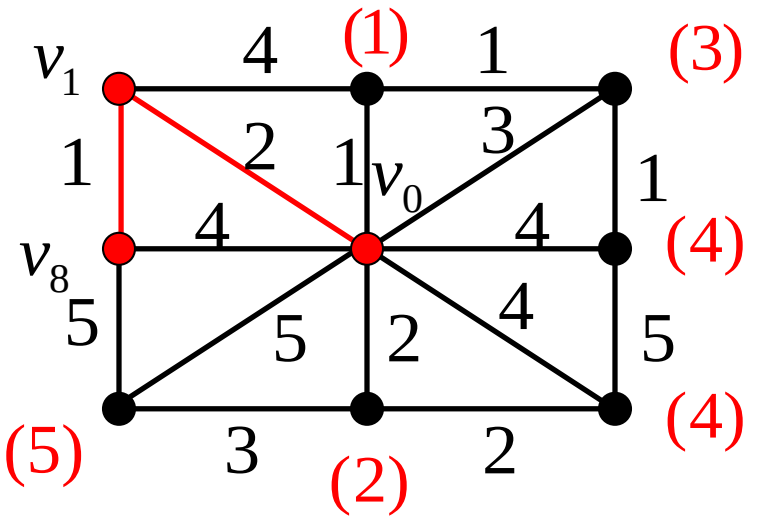
$\Rightarrow$



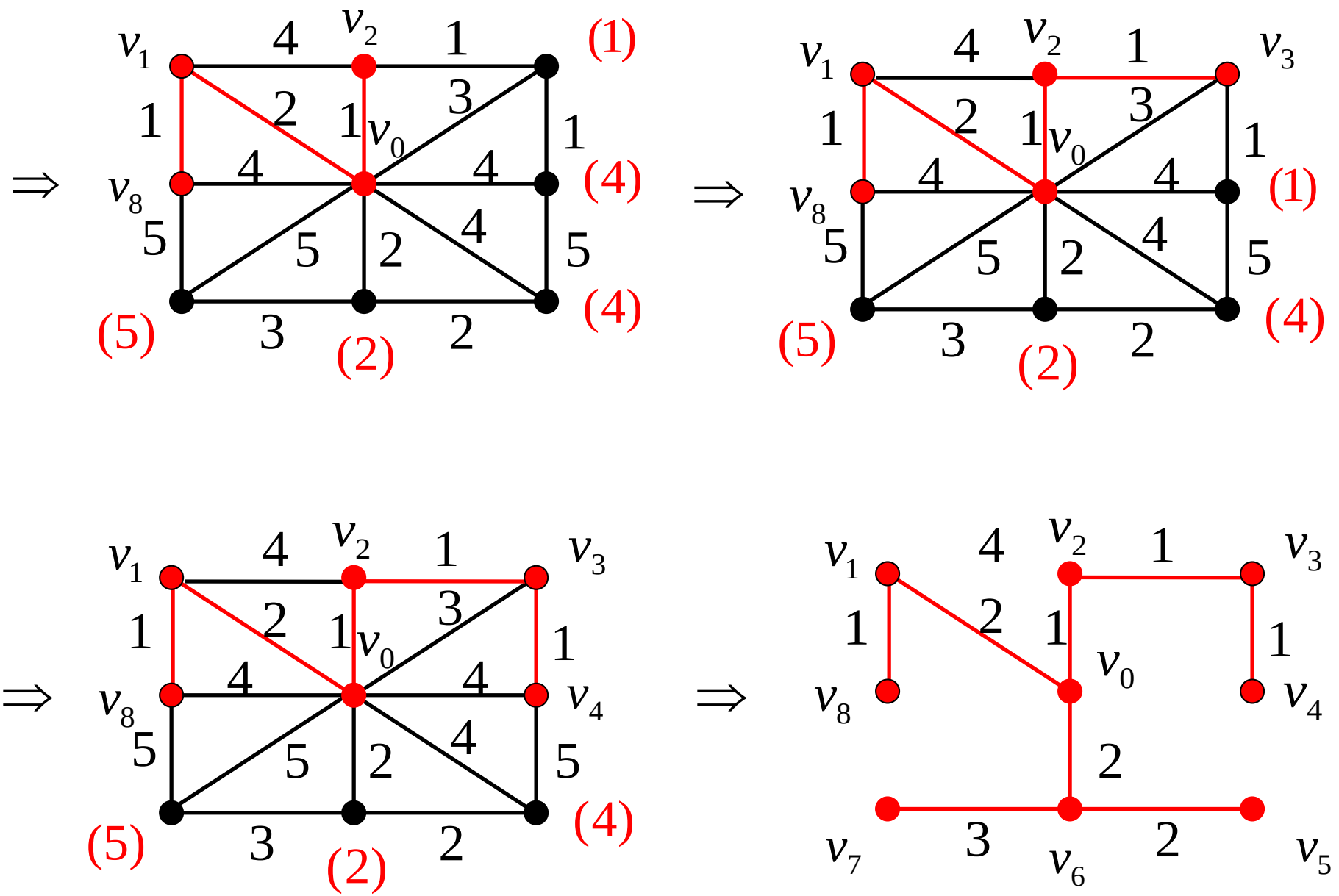
$\Rightarrow$



$\Rightarrow$



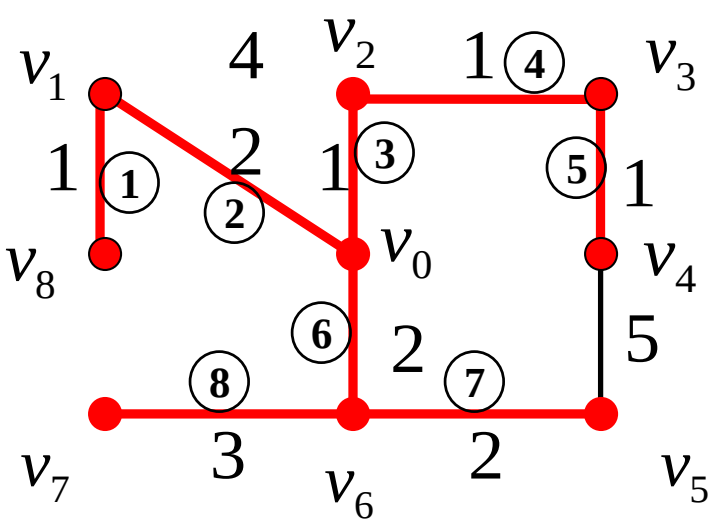
（每次只需修改和新增点相连的点的距离）


⇒
v1
v2
v0
v8
(1)
(4)
(5)
(2)
⇒
v3
⇒
v4
⇒
v7
v6
v5

Dijkstra算法生成的支撑树的性质

加入任何弦形成的回路中，弦的权值最大

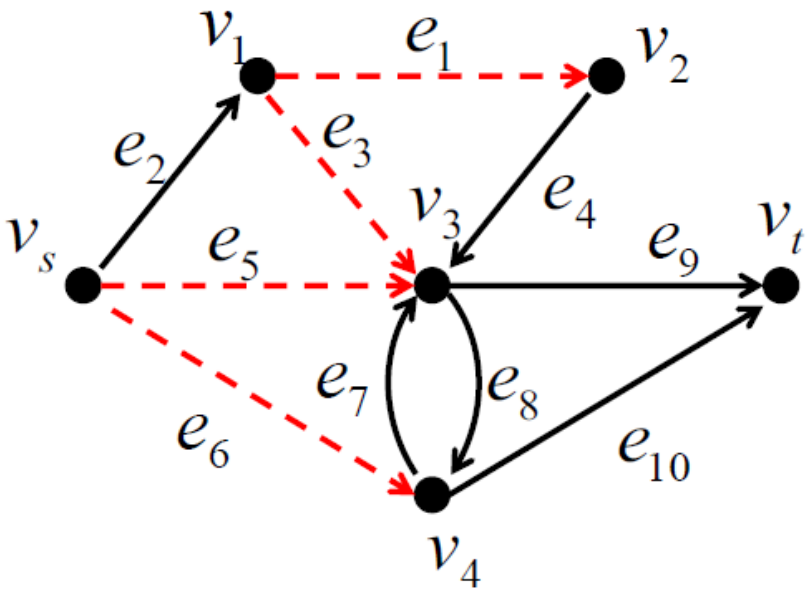
理由：以右图回路为例，其中圆中数字为进入树枝的顺序。首先容易看出，第 7 次进入回路的树枝权一定不大于弦的权，第 6 次进入的树枝同样，然后可看出，第 5 次进入的一定不大于第 6 次进入的，然后以此类推可知上述性质成立



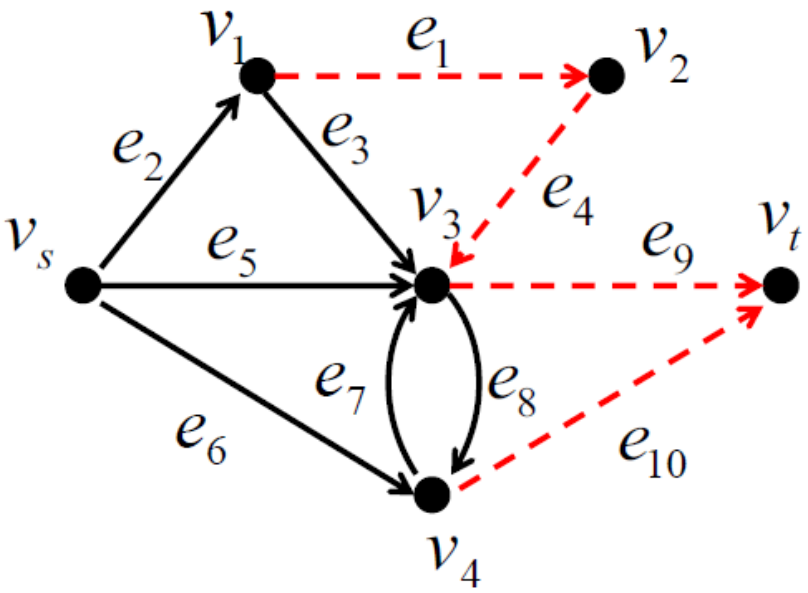
研究组合问题，递推，构造是常见的手法

边割 对于图 $G=(V,E)$，任取 $S\subset V$，其补集为 $\overline{S}$，若 S 和 $\overline{S}$ 都不是空集，称两个端点分属 S 和 $\overline{S}$ 的边的集合为 G 的一个**边割**，记为 $\{S,\overline{S}\}$

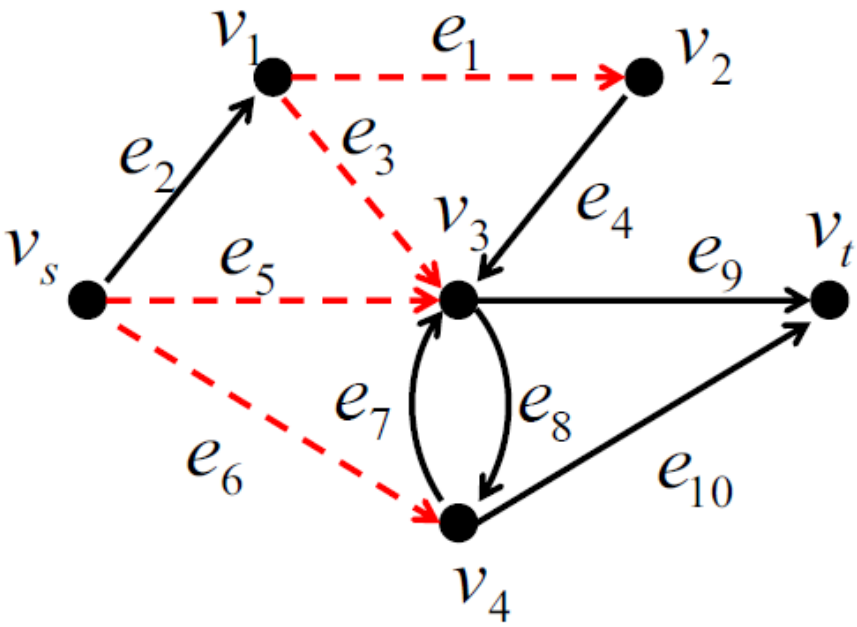
$\{e_1,e_3,e_5,e_6\}$

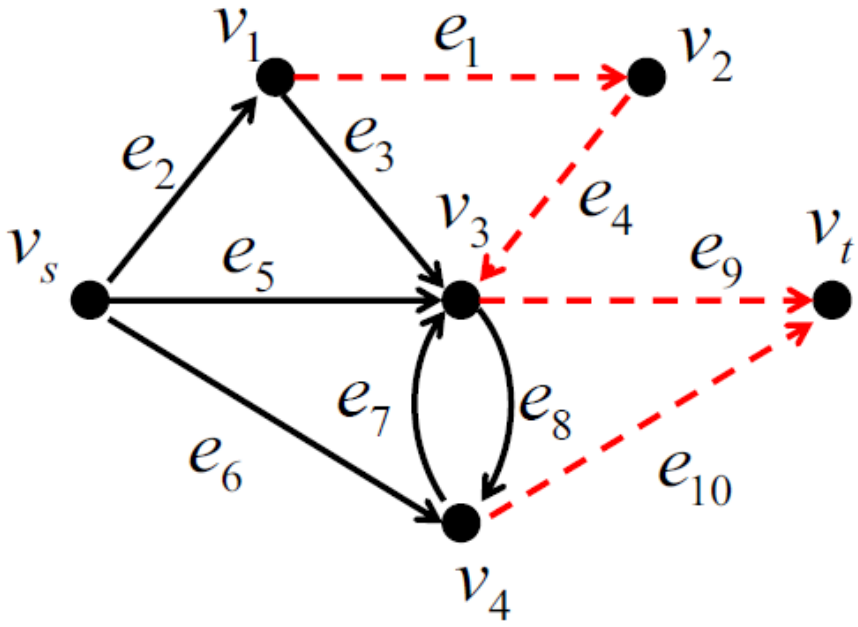


$\{e_1,e_4,e_9,e_{10}\}$



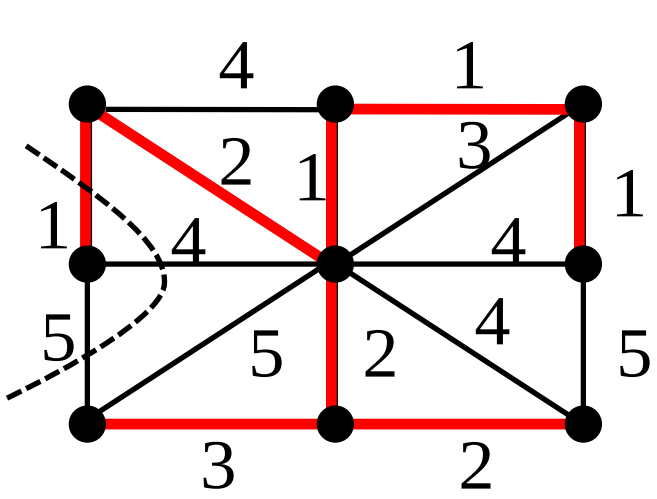
割集 当除去边割后，连通图变为不连通，而除去边割的真子集后，连通图仍然连通

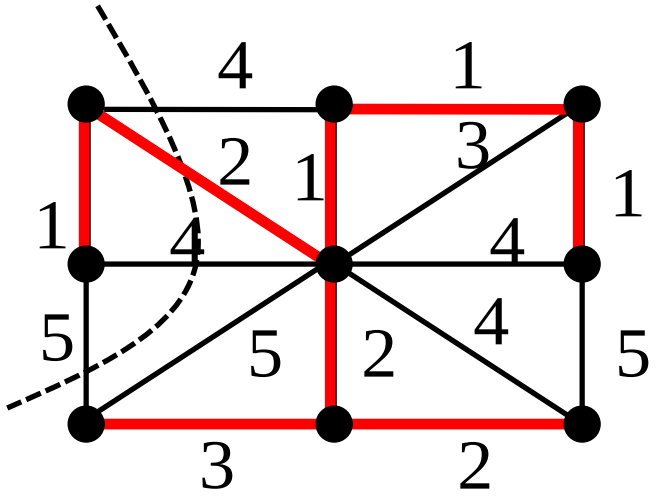


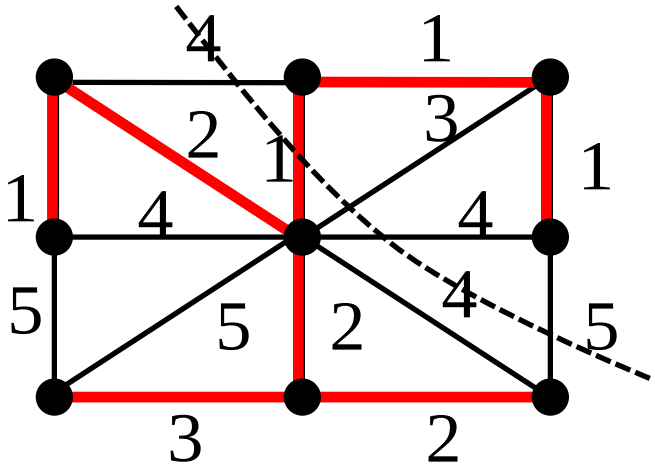


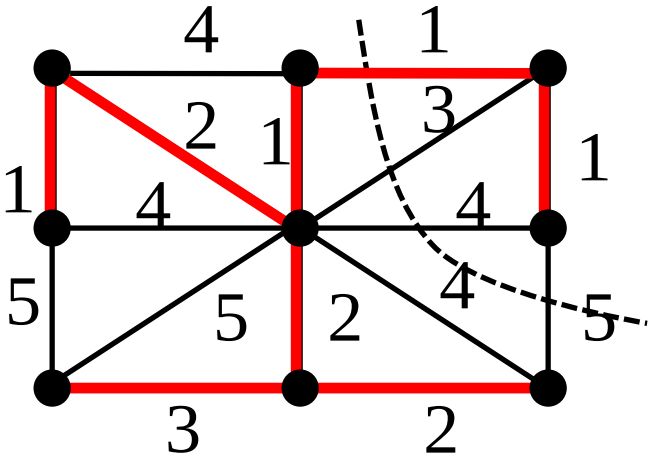
$\{e_1, e_3, e_5, e_6\}$是割集，$\{e_1, e_4, e_9, e_{10}\}$是边割，不是割集

教材采用的 Dijkstra 算法生成的支撑树的性质



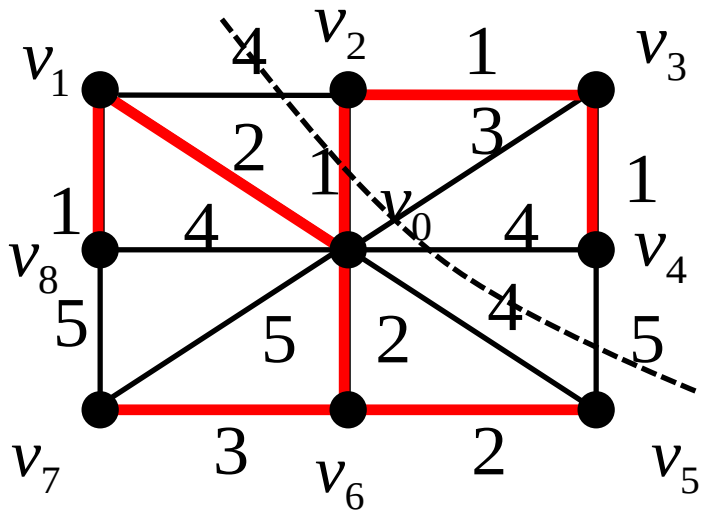




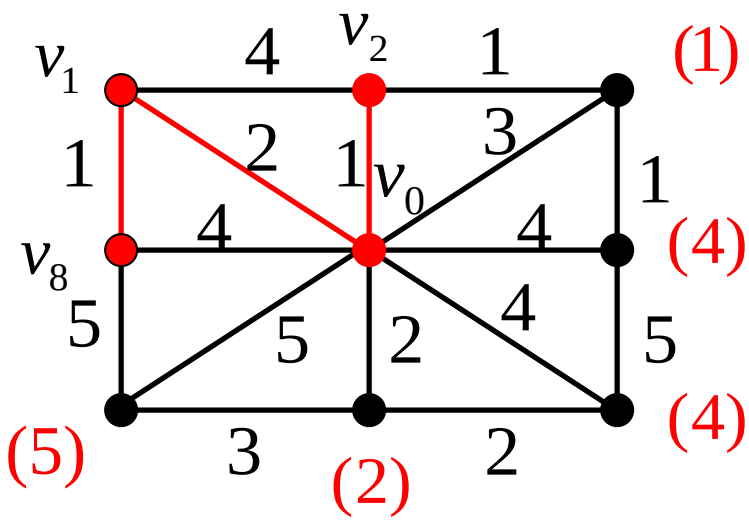


每个树枝是虚线所指出的*唯*一割集中的最短的边

如何比较右图 (v_0,v_2) 和 (v_4,v_5) ？



将 v_2 加入 S 时两者没有比较



不能直接得到“树枝最短”的性质！

利用“弦的权值最大的性质”可以得到上述性质

定理：*T* 是最小支撑树的充要条件是：任何树枝都是所在的唯一割集中权值最小的边

必要性：如果不是，用权值最小的边代替相应树枝可得总权值更小的支撑树

充分性：加入任何弦形成的回路中，弦和回路上任何树枝都在某个唯一的割集上，所以弦的权值最大

$\Rightarrow$ **Dijkstra算法产生的是最小支撑树**

算法复杂性： **Kruskal** $n^2 \log_2 n$ $>$ n^2 **Dijkstra**

原始prim算法

我们可以记一个数组d[]，d[i]表示第i个节点到树中任意一个节点的最小值；
最开始d[1]=0，跟1有边的点的d值为这条边的边权，其它的d值为正无穷；
然后把n个节点的d值比较，得到和树距离最小的点，将这个点加到树上，更新和这个点有边的节点的d值

这样的时间复杂度：因为每次只用比较n个d值，更新最多n个d值，所以时间复杂度是O(n(n+n))，也就是$O(n^2)$

https://www.zhihu.com/question/370649491/answer/1165243547

```
const int N = /* n的最大值 */ + 5;
vector< pair<int, int> > to[N]; //点的编号，边权
int d[N], done[N]; //d是上面说的d，done表示是否在树上
int main()
{for(int i = 1; i <= n - 1; i++) {
        int u = 1;
        for(int j = 2; j <= n; j++)
                if(done[u] || (!done[j] && d[j] < d[u]))
                        u = j; //找最小值
        done[u] = true;
        for(int i = 0; i < (int)to[u].size(); i++)
                if(d[to[u][i].first] > to[u][i].second)
                        d[to[u][i].first] = to[u][i].second; //更新d}
turn 0; //输出}
```

最短路算法

Floyd算法又称为插点法，是一种利用动态规划的思想寻找给定的加权图中多源点之间最短路径的算法，与Dijkstra算法类似。该算法名称以创始人之一、1978年图灵奖获得者、斯坦福大学计算机科学系教授罗伯特·弗洛伊德命名。

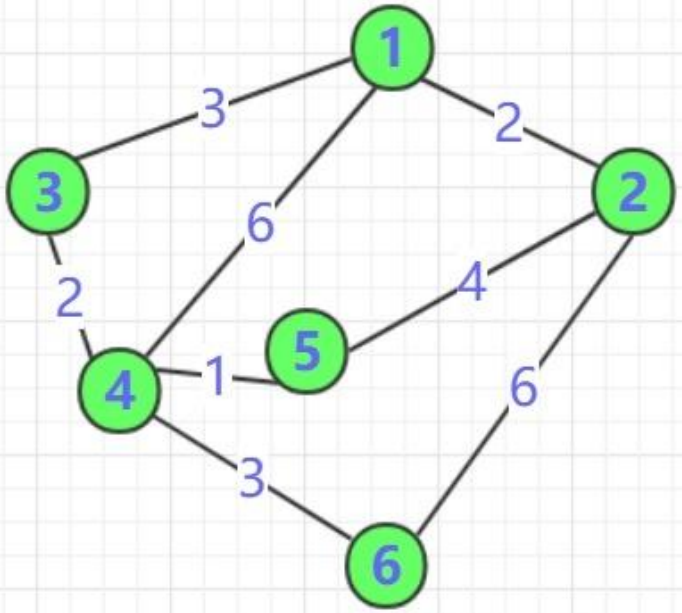
1. 邻接矩阵dist储存路径，同时最终状态代表点点的最短路径。如果没有直接相连的两点那么默认为一个很大的值(不要溢出)！而自己的长度为0.
2. 从第1个到第n个点依次加入图中。每个点加入进行试探是否有路径长度被更改。
3. 而上述试探具体方法为遍历图中每一个点(i,j双重循环)，判断每一个点对距离是否因为加入点而发生最小距离变化。其中状态转移方程为: dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j])
其中dp[x][y]的意思可以理解为x到y的最短路径。所以dp[i][k]的意思可以理解为i到k的最短路径dp[k][j]的意思可以理解为k到j的最短路径.
4. 如果发生改变，那么两点(i,j)距离就更改。
重复上述直到最后插点试探完成。

```
1  int d[MAXN][MAXN]; // d[u][v] 表示从u -> v的权值 不存在的时候为0
2  int V; // 顶点个数
3
4  void Floyd()
5  {
6      for(int k = 0;k < V;k++)
7          for(int i = 0;i < V;i++)
8              for(int j = 0;j < V;j++)
9                  d[i][j] = min(d[i][j],d[i][k]+d[k][j]);
10 }
```

可想而知：时间复杂度$O(V^3)$，空间复杂度$O(V^2)$

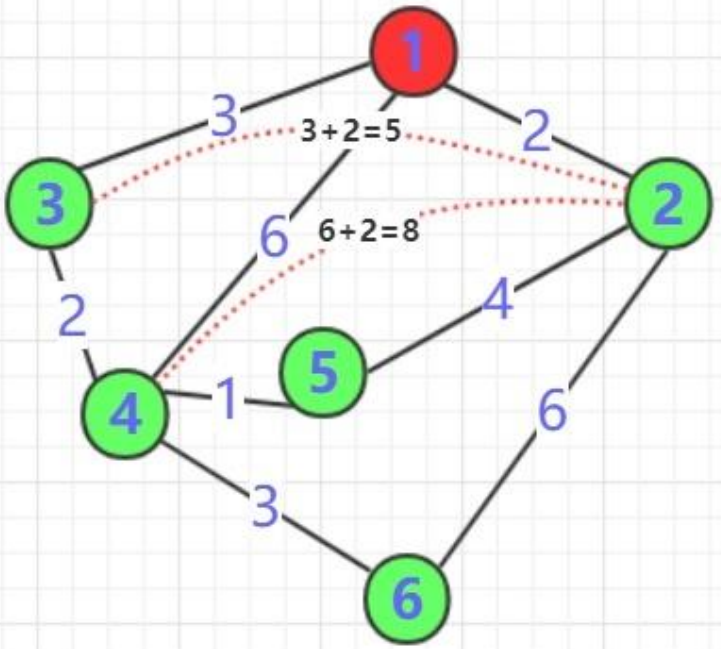
https://blog.csdn.net/qq_40953281/article/details/84889276

	1	2	3	4	5	6
1	0	2	3	6	∞	∞
2	2	0	∞	∞	4	6
3	3	∞	0	2	∞	∞
4	6	∞	2	0	1	3
5	∞	4	∞	1	0	∞
6	∞	6	∞	3	∞	0



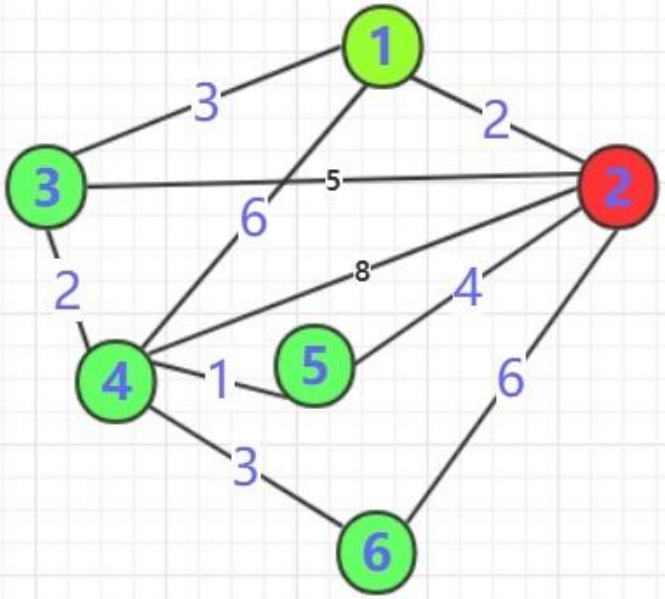
https://zhuanlan.zhihu.com/p/87480486

	1	2	3	4	5	6
1	0	2	3	6	∞	∞
2	2	0	5	8	4	6
3	3	5	0	2	∞	∞
4	6	8	2	0	1	3
5	∞	4	∞	1	0	∞
6	∞	6	∞	3	∞	0

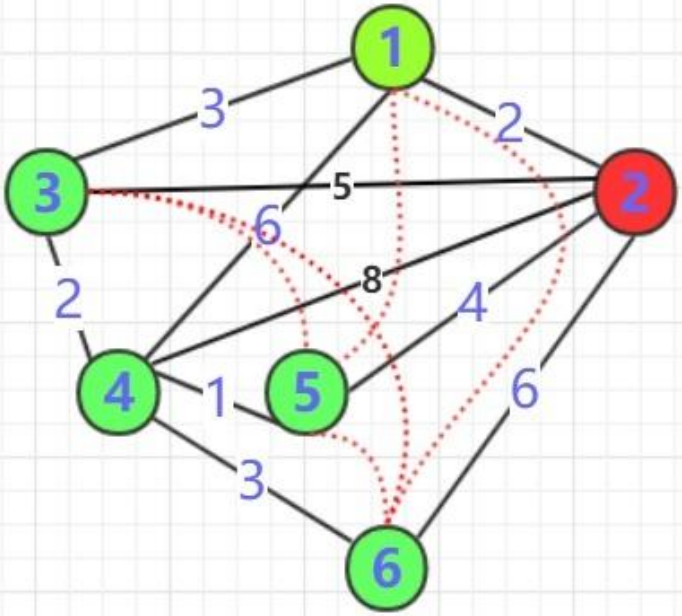


加入第一个节点1,大家可以发现，由于1的加入，使得本来不连通的2，3点对和2，4点对变得联通，并且加入1后距离为当前最小。(可以很直观加入5之后2，4，更短但是还没加入)。同时你可以发现加入1其中也使得3,1,4这样联通，但是3,1,4联通的话距离为9远远大于本来的(3,4)为2，所以不更新。

	1	2	3	4	5	6
1	0	2	3	6	∞	∞
2	2	0	5	8	4	6
3	3	5	0	2	∞	∞
4	6	8	2	0	1	3
5	∞	4	∞	1	0	∞
6	∞	6	∞	3	∞	0



	1	2	3	4	5	6
1	0	2	3	6	6	8
2	2	0	5	8	4	6
3	3	5	0	2	9	11
4	6	8	2	0	1	3
5	6	4	9	1	0	10
6	8	6	11	3	10	0



美国应用数学家Richard Bellman (理查德.贝尔曼, 动态规划的提出者)于1958 年发表了该单源点最短路径的算法。此外Lester Ford在1956年也发表了该算法。因此这个算法叫做Bellman-Ford算法。其实EdwardF. Moore在1957年也发表了同样的算法，所以这个算法也称为Bellman-Ford-Moore算法。

对所有的边进行n-1轮松弛操作，因为在一个含有n个顶点的图中，任意两点之间的最短路径最多包含n-1边。换句话说，第1轮在对所有的边进行松弛后，得到的是源点最多经过一条边到达其他顶点的最短距离；第2轮在对所有的边进行松弛后，得到的是源点最多经过两条边到达其他顶点的最短距离；第3轮在对所有的边进行松弛后，得到的是源点最多经过一条边到达其他顶点的最短距离

Bellman-ford 算法比dijkstra算法更具普遍性，因为它对边没有要求，可以处理负权边与负权回路。缺点是时间复杂度O(VE), V为顶点数，E为边数。

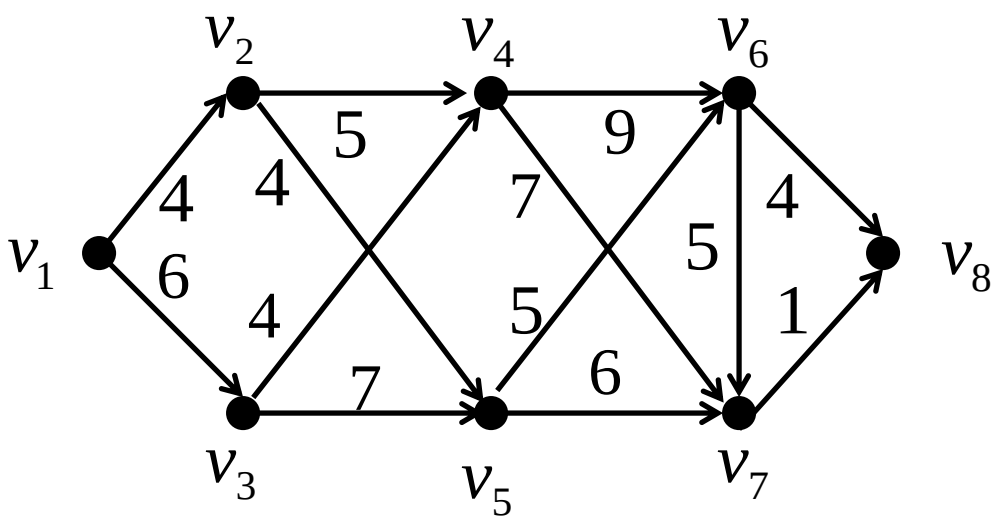
现在我们定义对点x, y的松弛操作是：
dist[y] = min(dist[y], dist[x] + e[x][y]);//这里的e[x][y]表示x、y之间的距离
松弛操作就相当于考察能否经由x点使起点到y点的距离变短。

Bellman-Ford算法告诉我们：

把所有边松弛一遍！

因为我们要求的是最小值，而多余的松弛操作不会使某个dist比最小值还小。所以多余的松弛操作不会影响结果。把所有边的端点松弛完一遍后，我们可以保证该轮松弛一定有边被松弛过了

例、 求 v_1 到 v_8 最短路



满足权值非负条件：$l_{ij} \geq 0, \forall i, j$

用值迭代公式

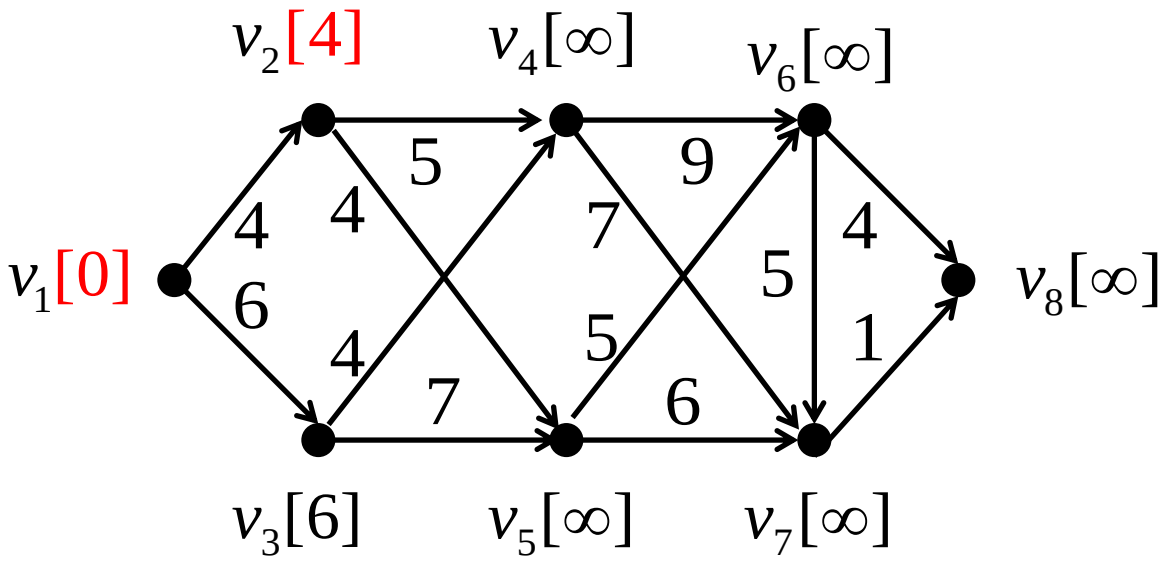
$$f_1(v_j) = l_{1j}, \ \forall j$$

$$f_{k+1}(v_j) = \min_i \{ f_k(v_i) + l_{ij} \}, \ \forall j$$

$f_k(v_j)$ 表示至多经过 $k-1$ 个中间点到达 v_j 的最短路

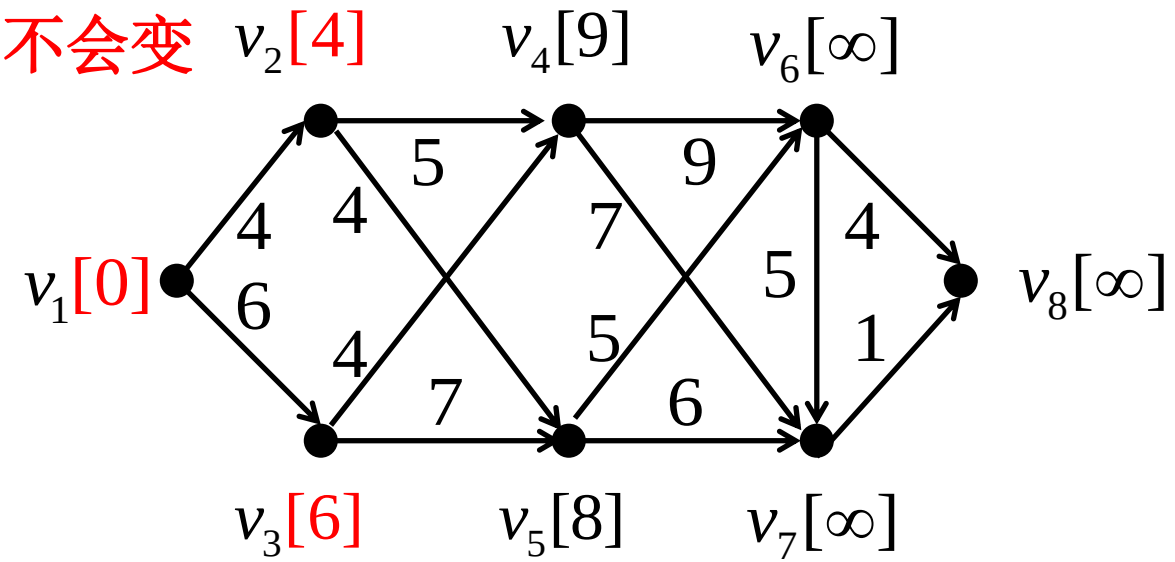
直接用值迭代法计算

$k=1$

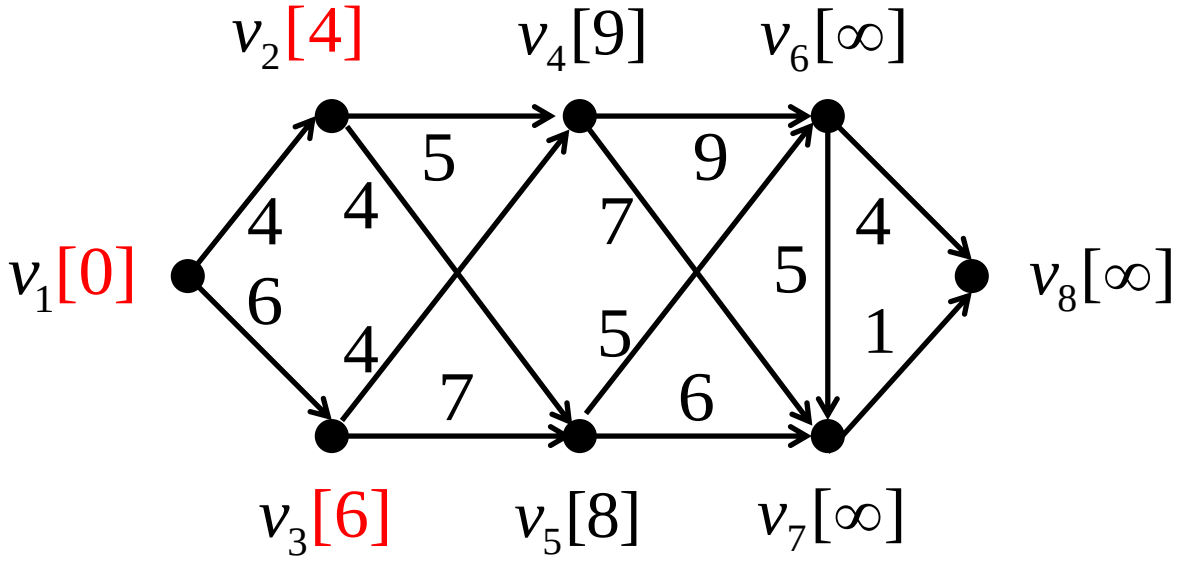


以后不会变

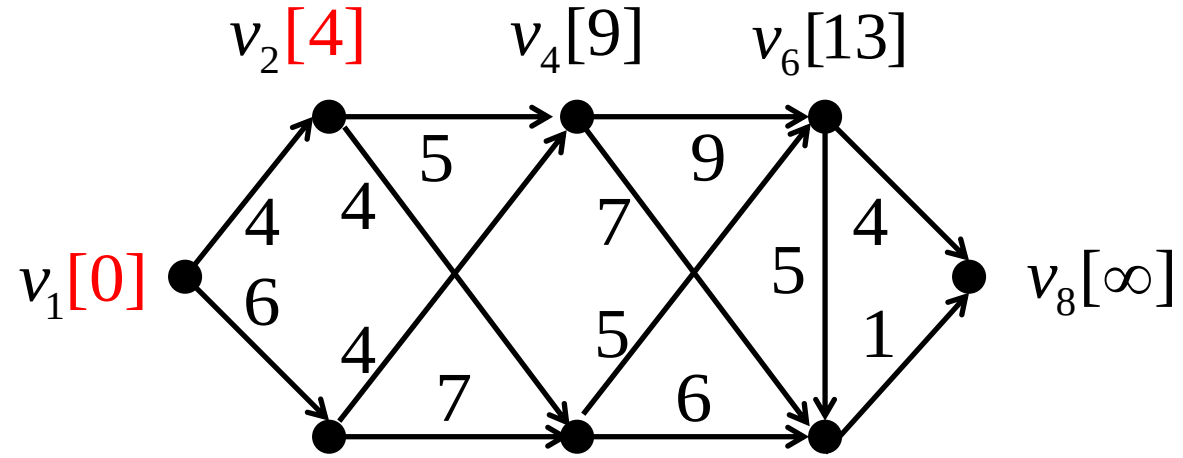
$k=2$



$k=2$

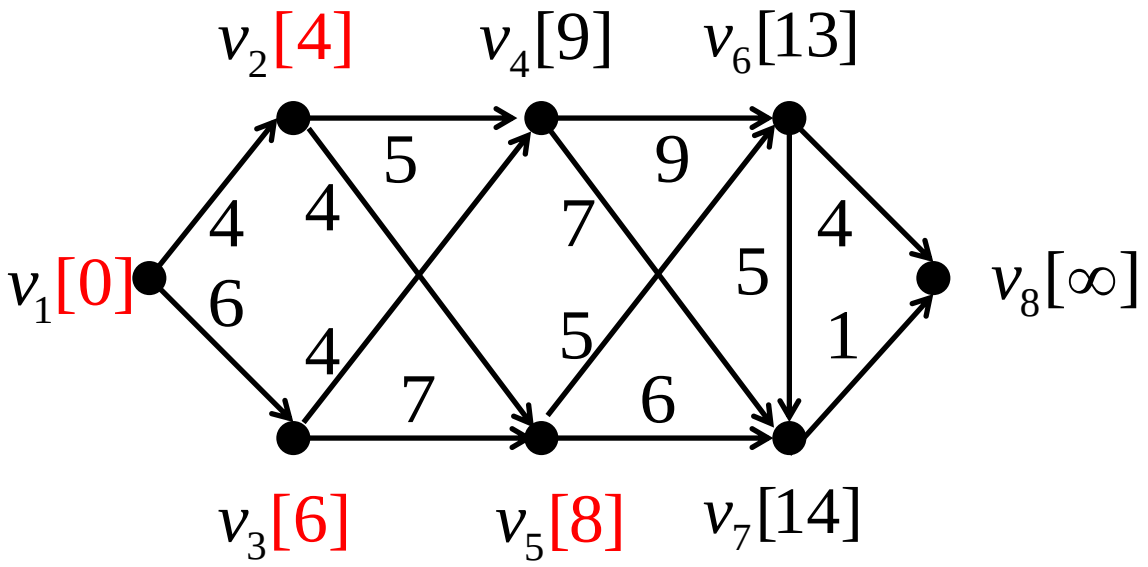

$v_2[4]$
$v_4[9]$
$v_6[\infty]$
$v_1[0]$
$v_8[\infty]$
$v_3[6]$
$v_5[8]$
$v_7[\infty]$
5
9
4
4
7
4
5
6
5
1
4
7
6

$k=3$

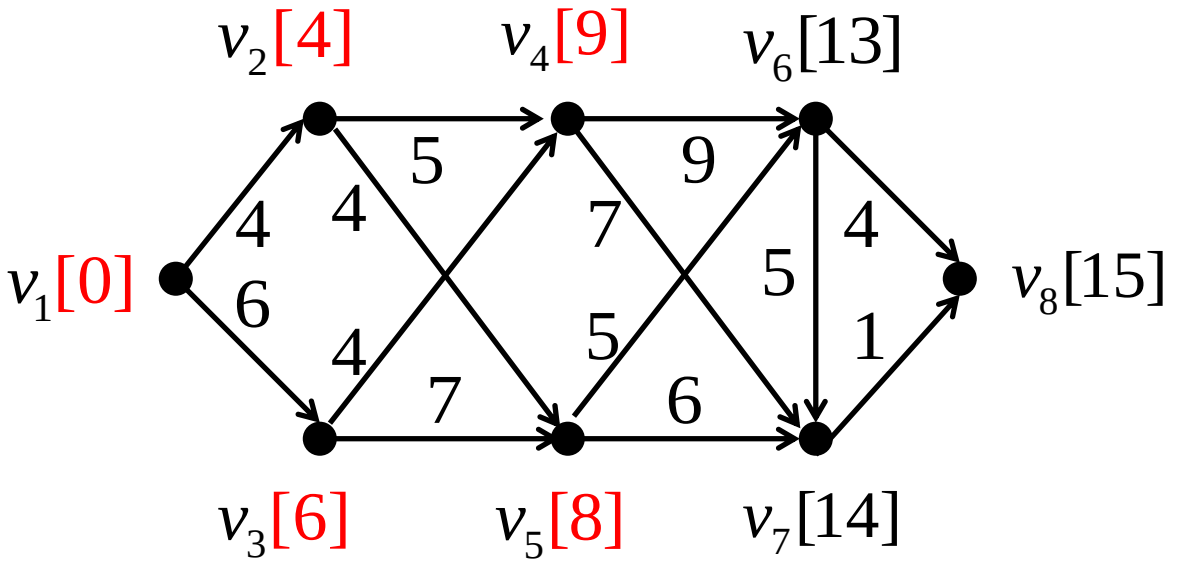

$v_2[4]$
$v_4[9]$
$v_6[13]$
$v_1[0]$
$v_8[\infty]$
$v_3[6]$
$v_5[8]$
$v_7[14]$
5
9
4
4
7
4
5
6
5
1
4
7
6

以后不会变

$k=3$

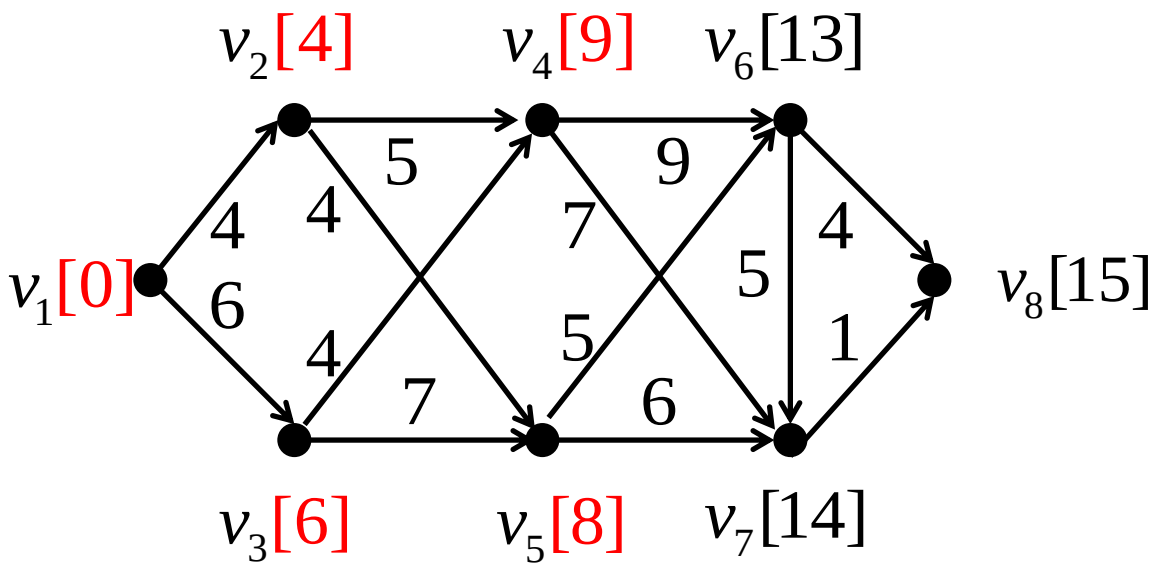

$v_2[4]$
$v_4[9]$
$v_6[13]$
$v_1[0]$
$v_8[\infty]$
$v_3[6]$
$v_5[8]$
$v_7[14]$
5
9
4
4
7
4
6
5
1
4
7
6

$k=4$


$v_2[4]$
$v_4[9]$
$v_6[13]$
$v_1[0]$
$v_8[15]$
$v_3[6]$
$v_5[8]$
$v_7[14]$
5
9
4
4
7
4
6
5
1
4
7
6

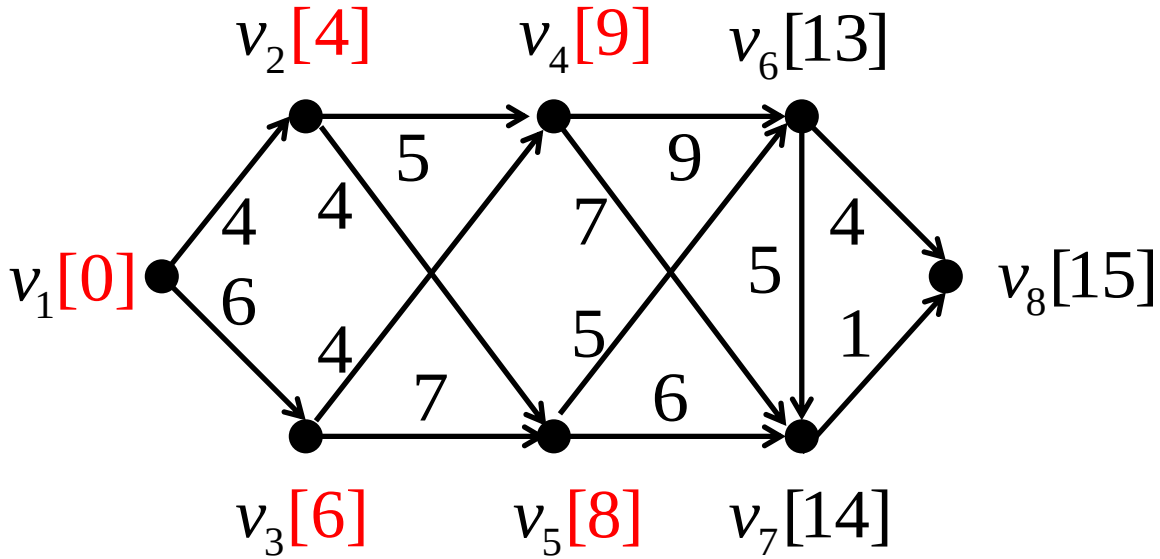
以后不会变

$k=4$



以后不会变

$k=5$



$f_5(v_j)=f_4(v_i),\ \forall j$ 停止

```
1  void Bellman_Ford(int s) // 不存在负圈的情况
2  {
3      for(int i = 0;i < V;i++) d[i] = INF;
4      d[s] = 0; // 到自己为0
5      for(int j = 0;j < V-1;j++)
6      {
7          for(int i = 0; i < E; i++)
8          {
9              Edge temp = edges[i];
10             d[temp.to] = min(d[temp.to],d[temp.from]+temp.cost)
11         }
12     }
13 }
```

如果图中不存在s可达的负圈，那么最短路不会经过一个顶点两次，也就是说 最多通过V-1条边，所以循环最多只会执行V-1次。

Bellman-Ford算法可以很简单处理负权环，只需要再多对每条边松弛一遍，如果这次还有点被更新，就说明存在负权环。

```cpp
void Bellman_Ford(int s)
{
    for(int i = 0;i < V;i++) d[i] = INF;
    d[s] = 0; // 到自己为0
    for(int j = 0;j < V;j++)
    {
        for(int i = 0; i < E; i++)
        {
            Edge temp = edges[i];
            if(d[temp.from] != INF && d[temp.to] > d[temp.from]+temp.cost)
            {
                d[temp.to] = d[temp.from] + temp.cost;
                // 只要再次加上到第V-1次的特判
                if(j == V-1)
                {
                    cout << "存在负圈" << endl;
                    return;
                }
            }
        }
    }
}
```

在Bellman算法中每一次都要全部遍历所有的边，而且如果d[i]本身不是最短路径

那么进行那个松弛操作之后的d[i]依然不是最短，所以可以对此进行优化：

- 找到最短路径已经确定的顶点，更新从他出发相邻顶点的最短距离
- 从此不需要在更新上面已经确定的哪些顶点（即不需要遍历）

Dijkstra算法的本质：在最开始时，只有起点的最短距离是确定的（而且所有点都未曾使用）。而在尚未使用的顶点中，距离d[i]最小的顶点就是最短距离已经确定的顶点。因为不存在负边，所以d[i]不会在以后的更新中变小。

不定期最短路问题的Dijkstra算法

一般性问题：

连通图 $G=(V,E)$ 各边 (v_i,v_j) 有权 l_{ij}（$l_{ij}=\infty$ 表示两点间无边），任意给定两点 v_s,v_t，求一条道路 μ，使它是从 v_s 到 v_t 的所有道路中总权最小的道路，即

$$L(\mu_*)=\min_{\mu\in\Omega_{st}} L(\mu) = \sum_{(v_i,v_j)\in\mu} l_{ij}$$

其中 Ω_{st} 表示从 v_s 到 v_t 的所有道路的集合

适用Dijkstra算法的问题：所有权值非负

非负权值带来的一个有利结果：

$$f_k(v_\alpha) = \min_{i \in I_k} f_k(v_i) \quad \Rightarrow \quad f_{k+1}(v_\alpha) = f_k(v_\alpha), \qquad \forall k, \alpha$$

I_k：最优函数值未固定的点集

每步迭代后未固定的最小的最优函数值不再发生变化

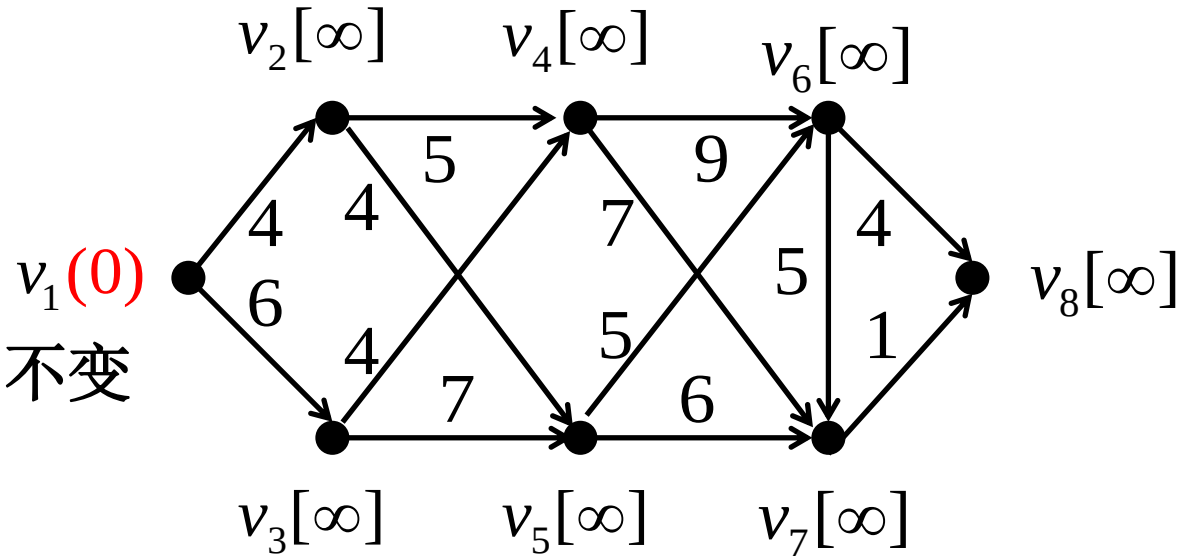
Dijkstra算法利用上述结果对值迭代法进行如下修改：

1）每次迭代后固定最小的最优函数值

2）每次迭代只利用刚固定函数值的点修改相邻的点

用Dijkstra算法计算

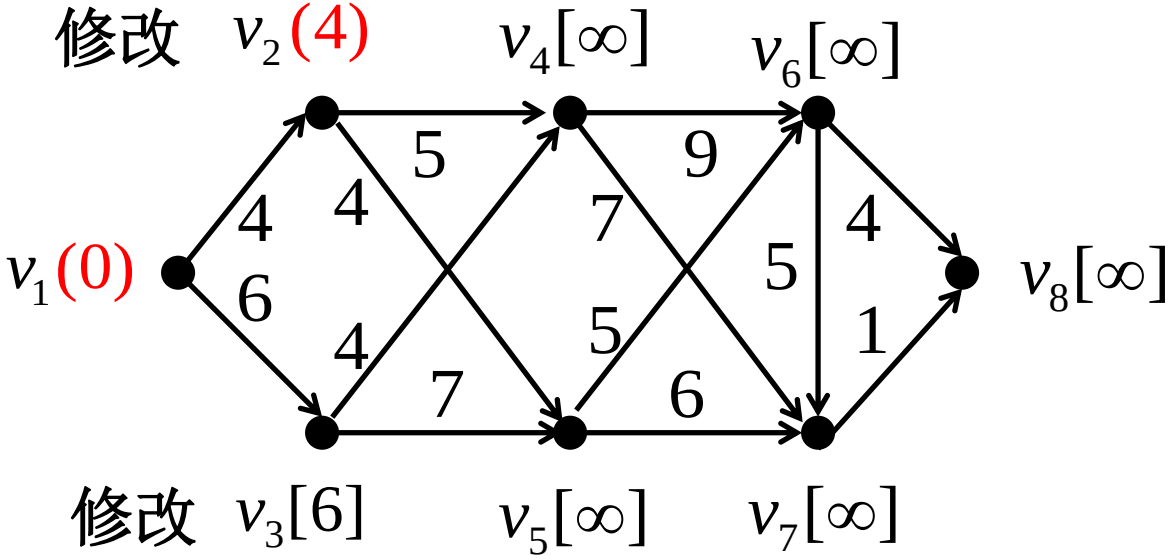
$k=0$

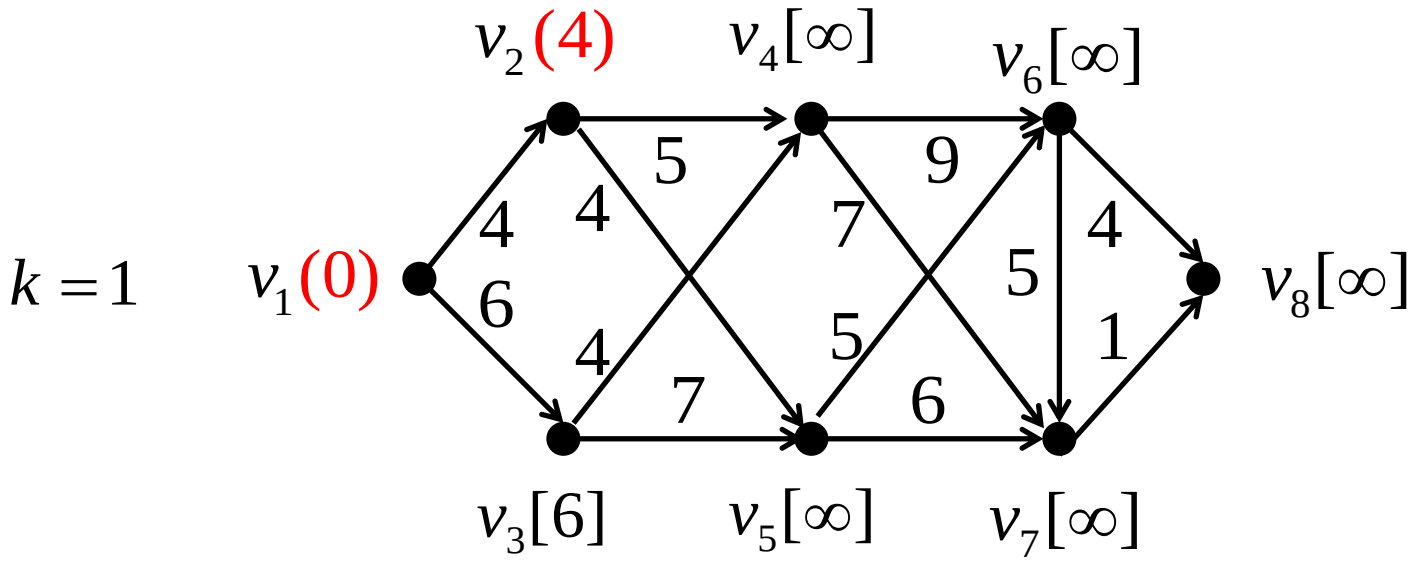


$k=1$

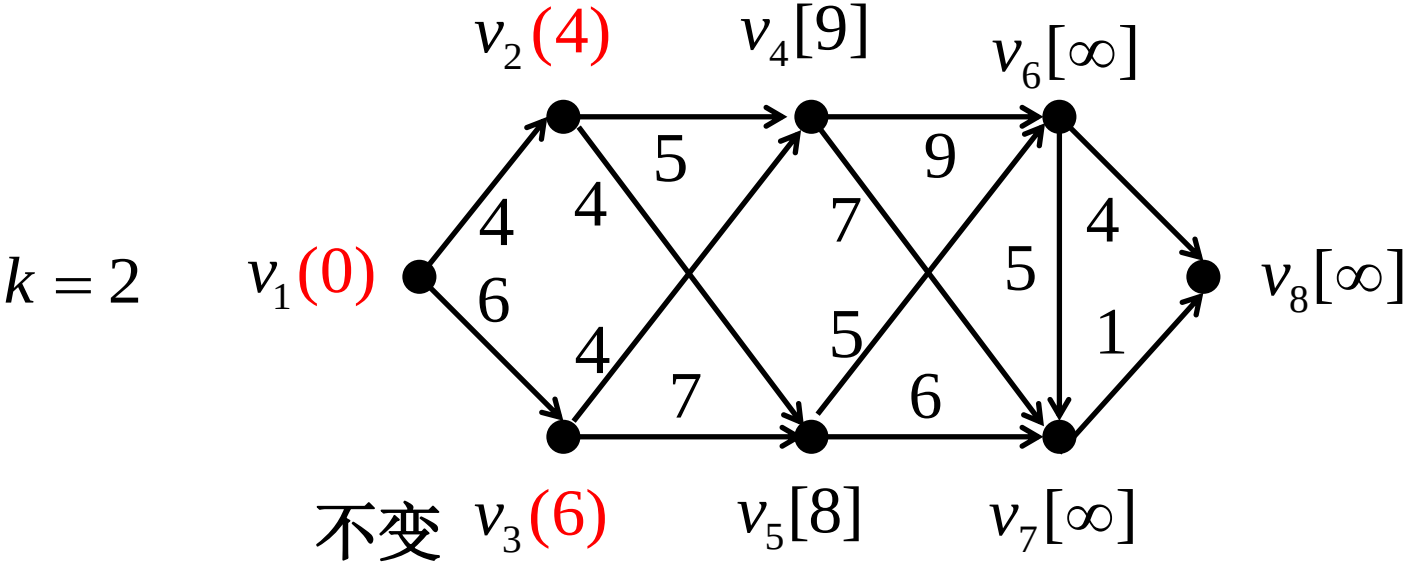
不变

修改 $v_2(4)$




$k=1$
$v_1(0)$
$v_2(4)$
$v_4[\infty]$
$v_6[\infty]$
$v_8[\infty]$
$v_3[6]$
$v_5[\infty]$
$v_7[\infty]$
4
6
5
4
4
7
7
5
9
6
5
4
1

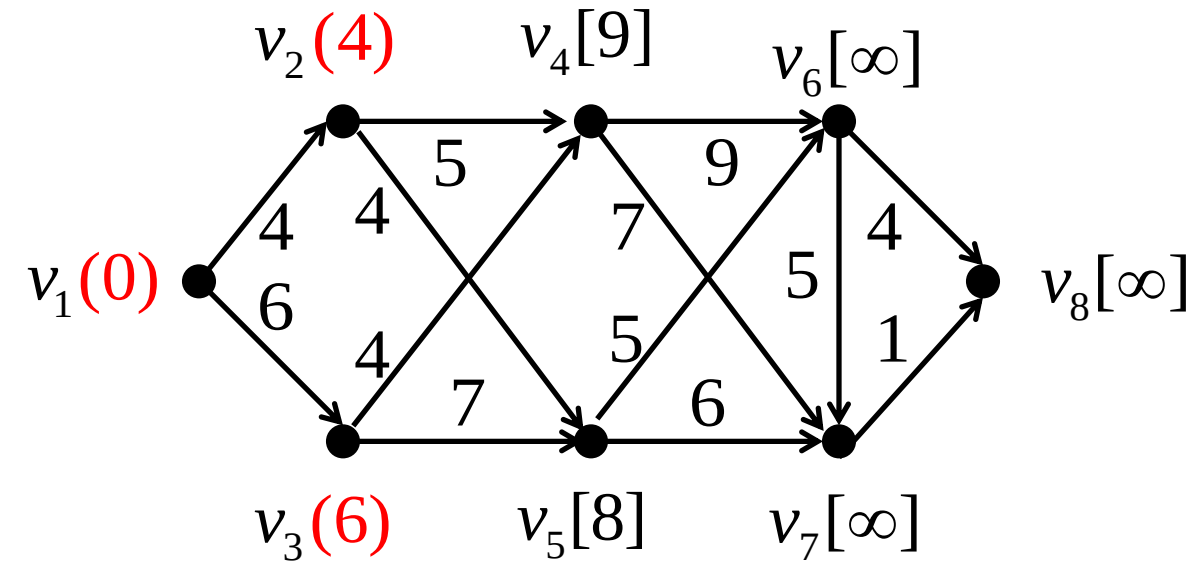
修改


$k=2$
$v_1(0)$
$v_2(4)$
$v_4[9]$
$v_6[\infty]$
$v_8[\infty]$
$v_3(6)$
$v_5[8]$
$v_7[\infty]$
4
6
5
4
4
7
7
5
9
6
5
4
1

不变

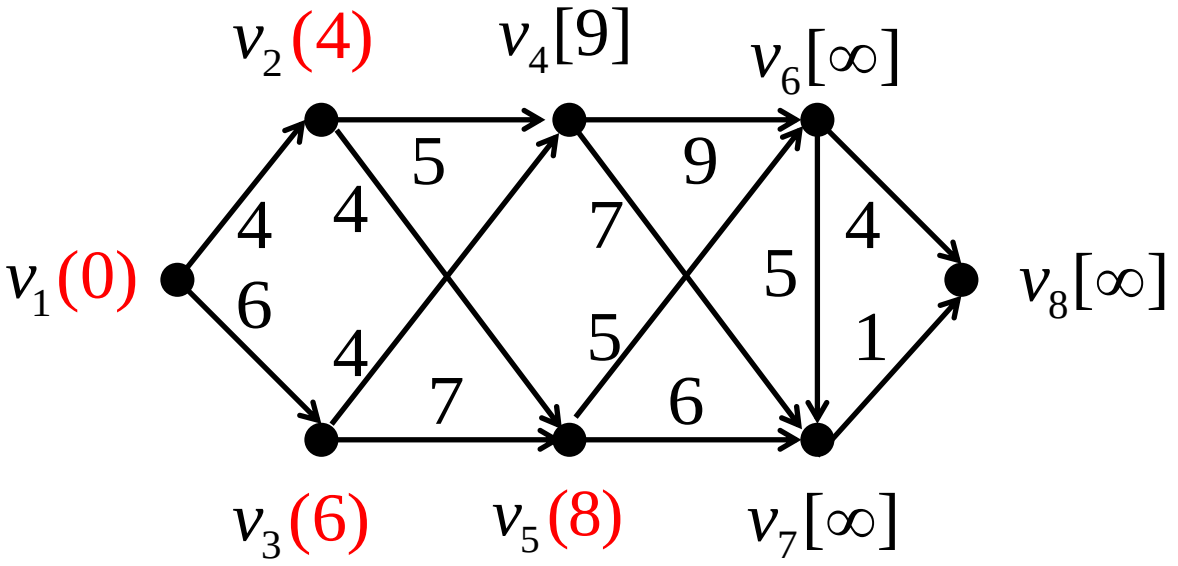
修改

$k=2$


$v_2(4)$
$v_4[9]$
$v_6[\infty]$
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5[8]$
$v_7[\infty]$
5
9
4
4
7
4
6
5
1
4
7
6

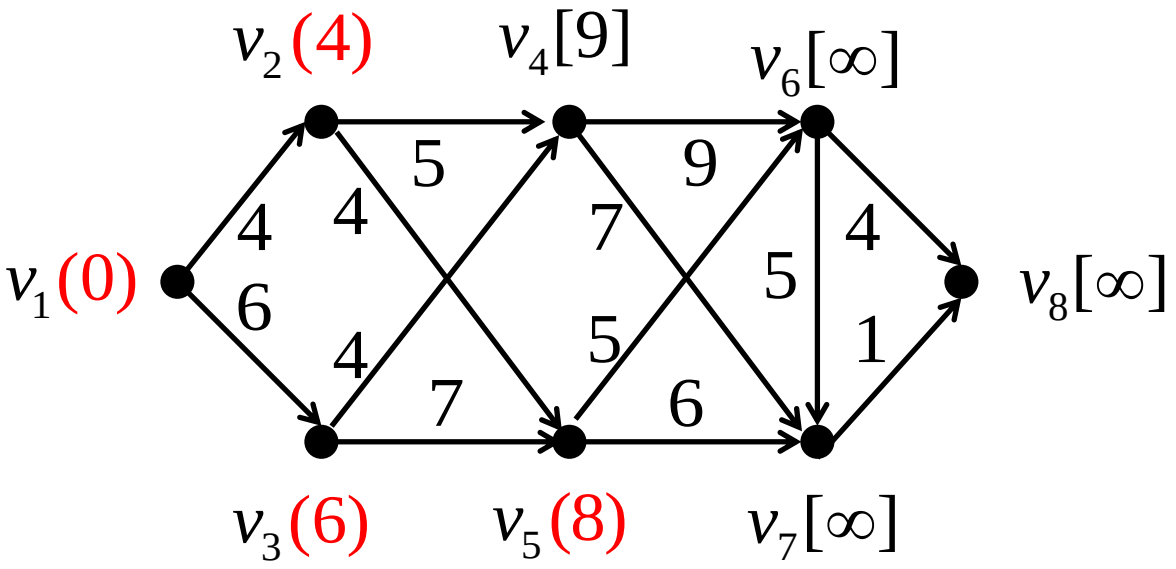
修改

$k=3$


$v_2(4)$
$v_4[9]$
$v_6[\infty]$
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[\infty]$
5
9
4
4
7
4
6
5
1
4
7
6

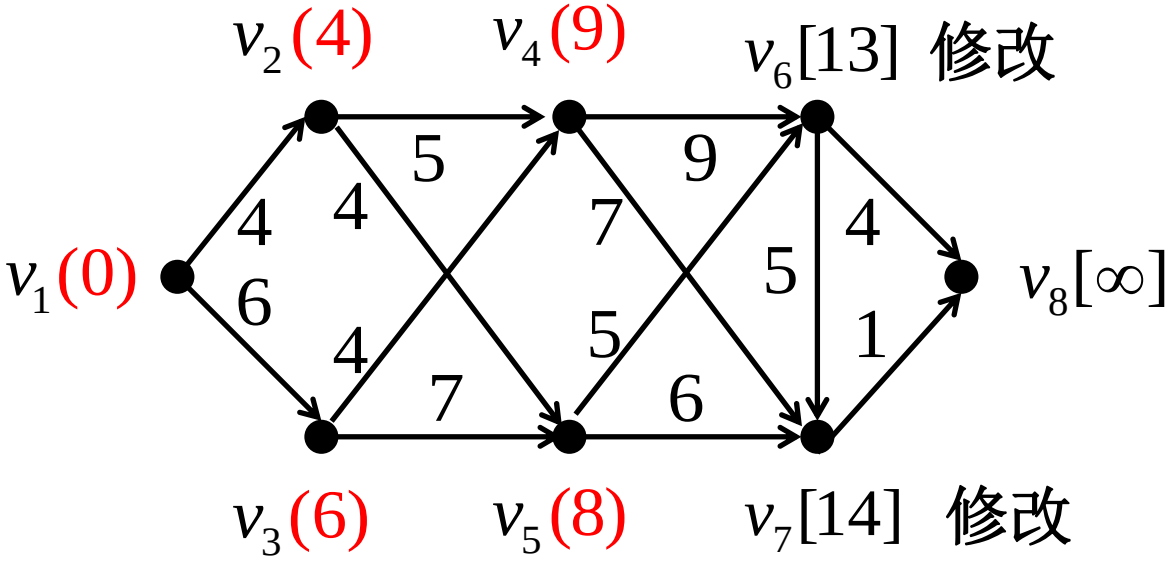
不变　修改

$k=3$

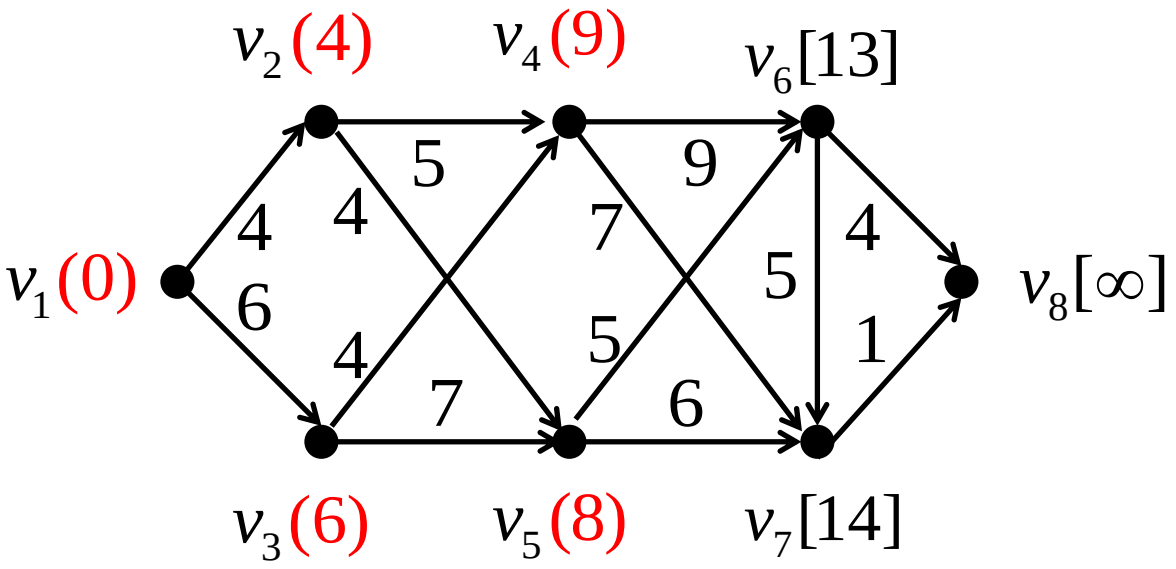

$v_2(4)$
$v_4[9]$
$v_6[\infty]$
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[\infty]$
5
9
4
4
7
4
6
5
5
1
4
7
6

不变

$k=4$

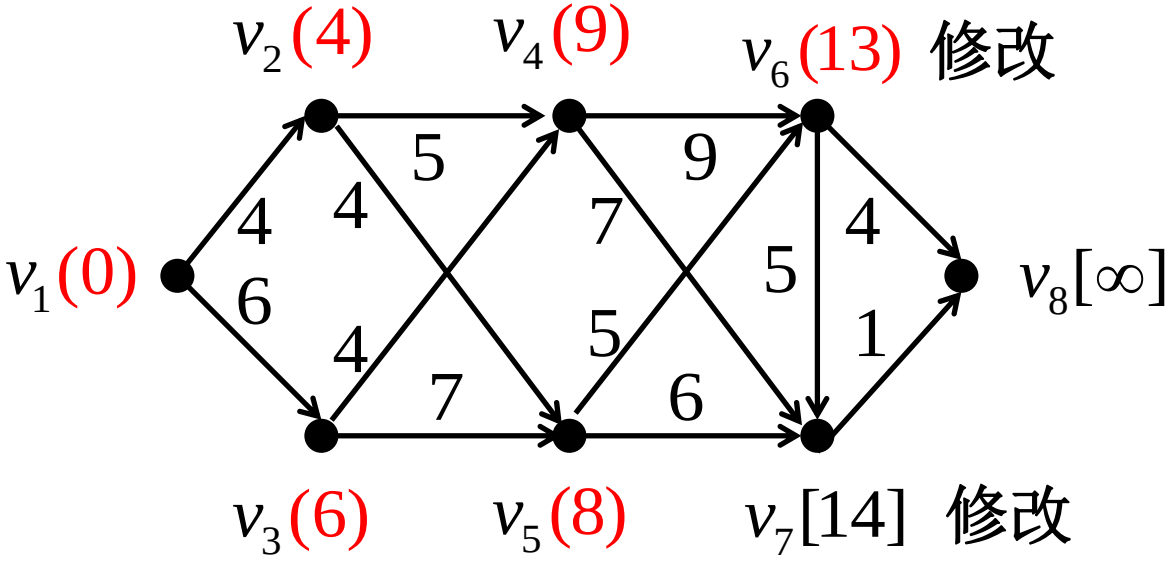

$v_2(4)$
$v_4(9)$
$v_6[13]$ 修改
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[14]$ 修改
5
9
4
4
7
4
6
5
5
1
4
7
6

$k=4$

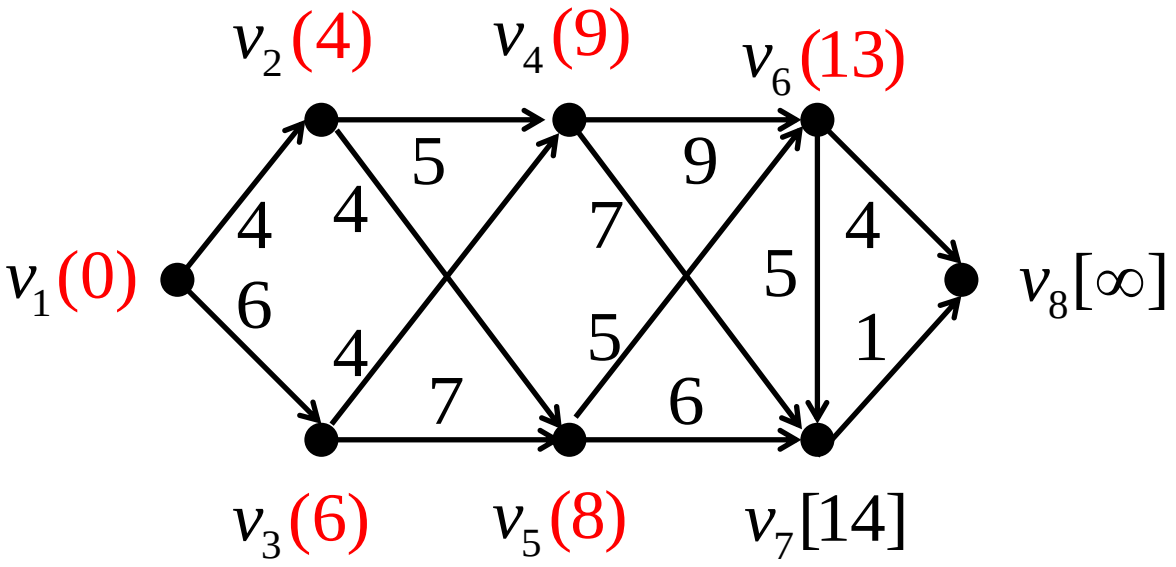

$v_2(4)$
$v_4(9)$
$v_6[13]$
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[14]$
5
9
4
4
7
4
6
5
1
4
7
6

不变

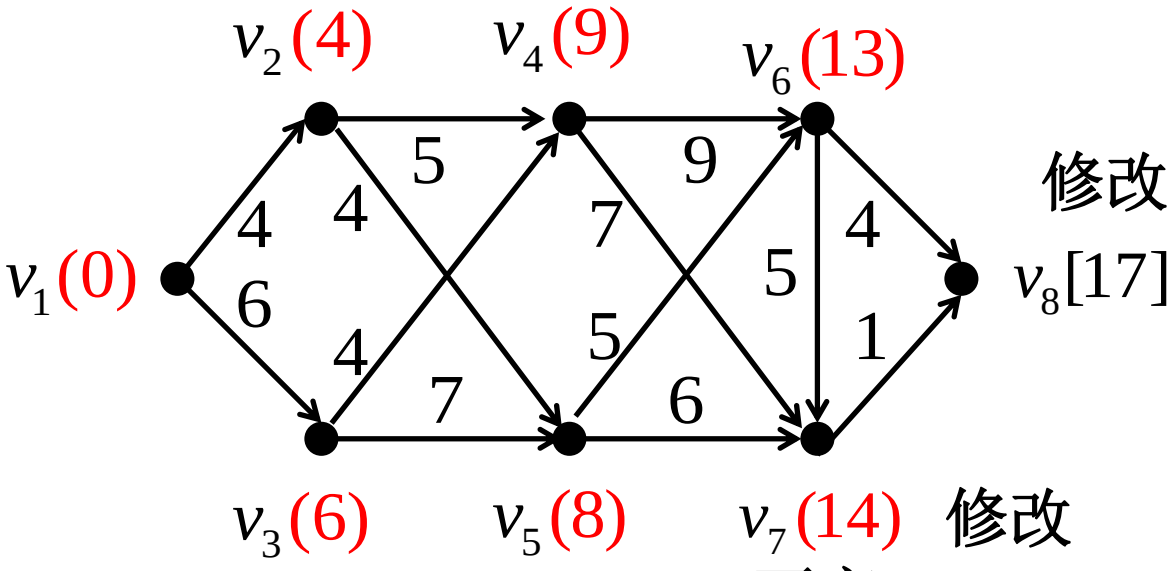
$k=5$


$v_2(4)$
$v_4(9)$
$v_6(13)$ 修改
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[14]$ 修改
5
9
4
4
7
4
6
5
1
4
7
6

$k=5$


$v_2(4)$
$v_4(9)$
$v_6(13)$
$v_1(0)$
$v_8[\infty]$
$v_3(6)$
$v_5(8)$
$v_7[14]$
5
9
4
4
7
4
6
5
1
4
7
6

$k=6$

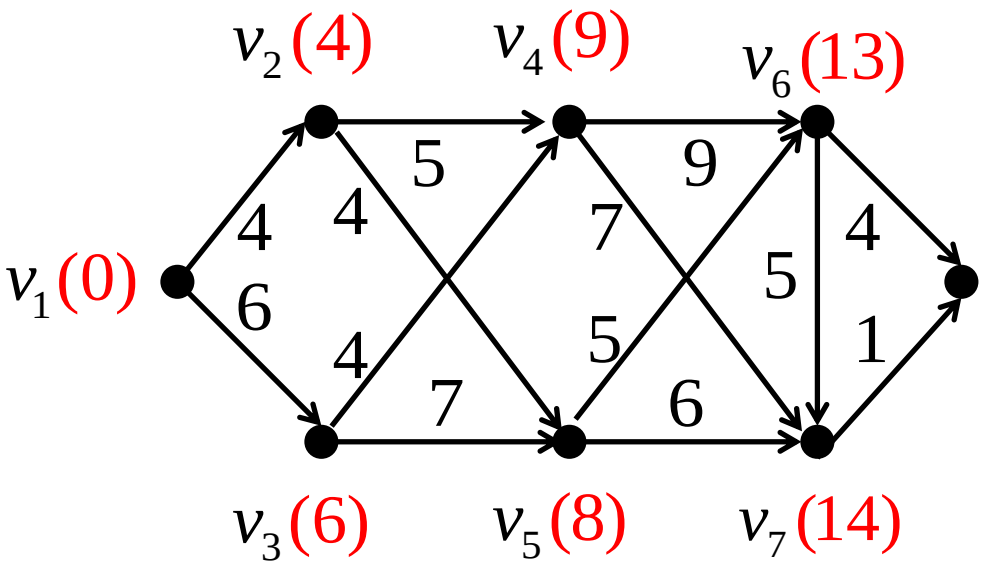

$v_2(4)$
$v_4(9)$
$v_6(13)$
$v_1(0)$
$v_8[17]$
$v_3(6)$
$v_5(8)$
$v_7(14)$
5
9
4
4
7
4
6
5
1
4
7
6

修改

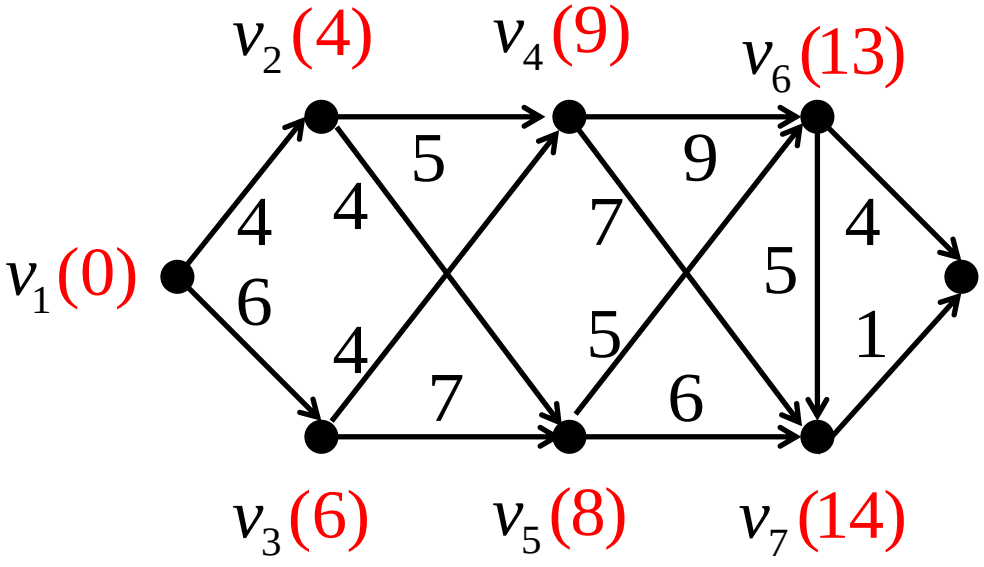
修改

不变

$k=6$


$v_2(4)$
$v_4(9)$
$v_6(13)$
$v_1(0)$
$v_8[17]$
$v_3(6)$
$v_5(8)$
$v_7(14)$
5
9
4
4
7
4
6
5
1
4
7
6

$k=6$


$v_2(4)$
$v_4(9)$
$v_6(13)$
$v_1(0)$
$v_8(15)$
$v_3(6)$
$v_5(8)$
$v_7(14)$
5
9
4
4
7
4
6
5
1
4
7
6

修改

不变

```cpp
int cost[MAXN][MAXN]; // cost[i][j] 表示从i到j之间的权值（不存在是为INF）
int d[MAXN]; // 从起点到其他点的最短距离
bool used[MAXN]; // 已经使用过的图（已经确定最短距离的点）
int V; // 点的个数

void Dijkstra(int s)
{
    fill(d,d+V,INF); // algorithm中的函数 将d数组全部赋为INF
    fill(used,used+V,false);
    d[s] = 0;

    while(true)
    {
        int v = -1;
        // c从未使用过的点集中取一个距离最小的点
        for(int u = 0;u < V;u++)
            if(!used[u] && (v == -1 || d[u] < d[v])) v = u;
        if(v == -1) break; // 所有的点的最短路径确定则退出
        used[v] = true;
        for(int u = 0;u < V;u++)
        {
            d[u] = min(d[u],d[v]+cost[v][u]);
        }
    }
}
```

上面有一个操作是找到距离最小的点和标记是否使用，这个就可以使用堆来优化。上面代码的时间复杂度是 O(V2) , 我们可以通过堆（优先队列）降为 O(E*log(V))

实现

1. 将源点加入堆，并调整堆。
2. 选出堆顶元素u（即代价最小的元素），从堆中删除，并对堆进行调整。
3. 处理与u相邻的，未被访问过的，满足三角不等式的顶点

1):若该点在堆里，更新距离，并调整该元素在堆中的位置。

2):若该点不在堆里，加入堆，更新堆。

4. 若取到的u为终点，结束算法；否则重复步骤2、3。

```cpp
typedef pair<int,int> P; // first 是最短距离 second 是顶点编号
struct edge{int to, cost};
vector<edge> G[MAXN]; // 使用邻接表存图
int d[MAXN]; // 从起点到其他点的最短距离
bool used[MAXN]; // 已经使用过的图（已经确定最短距离的点）
int V; // 点的个数

void Dijkstra(int s)
{
    priority_queue<P,vector<P>, greater<P> > que; // 定义一个堆 从按最短距离小到的大排
    fill(d,d+V,INF);
    d[s] = 0;
    que.push(P(0,s));
    while(!que.empty()) // 为空就说明所有节点都已经用过
    {
        P temp = que.top(); que.pop();
        int v = temp.second;
        if(d[v] < temp.first) continue; // 没必要更新了
        for(int i = 0;i < G[v].size();i++)
        {
            edge e = G[v][i];
            if(d[e.to] > d[v]+e.cost)
            {
                d[e.to] = d[v]+e.cost;
                que.push(P(d[e.to],e.to));
            }
        }
    }
}
```

基于集合的写法（适用于稠密图）复杂度
初始化 距离数组 和 未用过的顶点的集合 均为 O (V)
外层循环，每次删除一个顶点，共 V个顶点，故为 O (V)
内层 Extrac-min 为 O (V)，遍历邻居为 O (N)，N 为每个节点最大的邻居数
综上，复杂度为 O (V) + O (V^2) + O (VN) ~ O (V^2)

基于优先队列的写法（适用于稀疏图）复杂度
初始化 距离数组 为 O (V)，初始化 未用过的顶点的集合为 O (VlogV)
外层循环 O (V)
内层循环遍历所有邻居为 O (N)，更新邻居为 O(log(V))
综上，总复杂度为 O (VNlogV) 或写为 O (ElogV)

当图稀疏时，V^2 ≫ E，后者更优；当图稠密时，V^2 ~ E，前者更优