第二讲 图的基本概念

§1.1 基本概念

1. **图(graph)**：图是一个有序二元组(V,E)，其中集合V称为顶点集，集合E是V中元素组成的某些无序对的集合，称为边集。

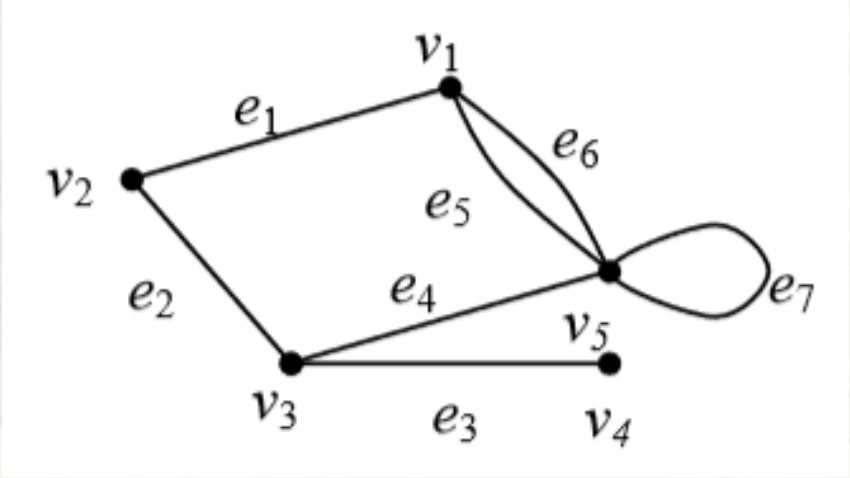
例1.1：G=(V,E)是一个图,其中

$$V=\{v_1,v_2,v_3,v_4,v_5\},$$

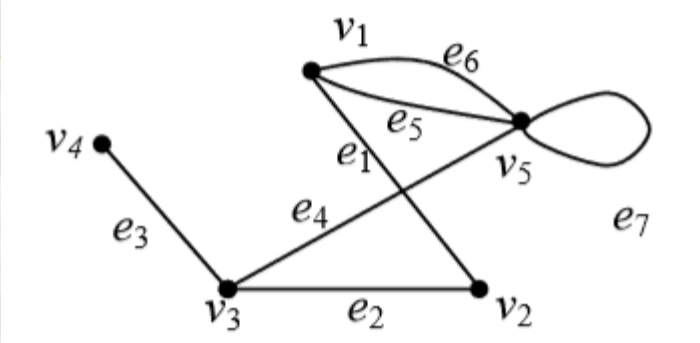
$$E=\{(v_1,v_2),(v_2,v_3),(v_3,v_4),(v_3,v_5),(v_1,v_5),(v_1,v_5),(v_5,v_5)\}。$$

2. 图的图示

通常，图的顶点可用平面上的一个点来表示，边可用平面上的线段来表示（直的或曲的）。这样画出的平面图形称为图的图示。 例如，例 1.1 中图的一个图示为



注：（1）由于表示顶点的平面点的位置的任意性，同一个图可以画出形状迥异的很多图示。比如下图是例1.1中图的另一个图示：



（2）图的图示直观易懂，因此以后一般说到一个图，总是画出它的一个图示来表示。

3. 一些术语和概念

设G=(V,E)是一个图，下述概念中顶点均取自V，边均取自E。

（1）点与边的关联(incident)：如果在图G中点v是边e的一个端点，则称点v与边e在图G中相关联。

（2）点与点的相邻(adjacent)：如果图上两点u,v被同一条边相连，则称u,v在图G中相邻。

（3）边与边的相邻：如果图G中两条边有至少一个公共端点，则称这两条边在图G中相邻。

（4）环边(loop)：图中两端点重合的边称为环边。

（5）重边(multiedge)：设u和v是图G的顶点，图G中连接u和v的两条或两条以上的边称为图G中u、v间的重边。

（6）简单图(simple graph)：既无环边也无重边的图称为简单图。

（7）完全图(complete graph)：任意两点间都有一条边的简单图称为完全图，n阶完全图记为K_n.

（8）平凡图(trivial graph)：只有一个顶点，没有边的图。

（9）空图(empty graph)：边集为空的图。

（10）零图(null graph)：顶点集为空的图。

（11）顶点v的度(degree)：图G中顶点v所关联的边的数目（环边计两次）称为顶点v的度，记为$d_G(v)$或$d(v)$。

（12）图G的最大度：$\Delta(G)=\max\{d_G(v)|v\in V(G)\}$；

图G的最小度：$\delta(G)=\min\{d_G(v)|v\in V(G)\}$。

（13）正则图(regular graph)：每个顶点的度都相等的图。

（14）图的补图(complement)：设G是一个图，以V(G)为顶点集，以$\{(x,y)|(x,y)\notin E(G)\}$为边集的图称为G的补图，记为$\bar{G}$。

定理 1.1.1 对任何图G，都有 $\sum_{v\in V(G)} d(v) = 2\varepsilon$.

4. 子图

子图(subgraph)：对图G和H，如果$V(H) \subseteq V(G)$且$E(H) \subseteq E(G)$，则称图 H 是图 G 的子图，记为$H \subseteq$G。

生成子图(spanning subgraph)：若H是G的子图且$V(H)=V(G)$，则称H是G的生成子图。

点导出子图(induced subgraph)：设G是一个图，$V' \subseteq V(G)$。以V'为顶点集，以G中两端点均属于V'的所有边作为边集所组成的子图，称为 G 的由顶点集V'导出的子图，简称为G的点导出子图，记为G[V']。

边导出子图(edge-induced subgraph)：设G是一个图，$E' \subseteq E(G)$。以E'为边集，以E'中边的所有端点作为顶点集所组成的子图，称为G的由边集E'导出的子图，简称为G的边导出子图，记为G[E']。

5. 路和圈

途径(walk)：图G中一个点边交替出现的序列$w = v_{i_0}e_{i_1}v_{i_1}e_{i_2}\cdots e_{i_k}v_{i_k}$称为图G的一条途径，其中$v_{i_0}$、$v_{i_k}$分别称为途径$w$的起点和终点，$w$上其余顶点称为中途点。

迹(trail)：图G中边不重复的途径称为迹。

路(path)：图G中顶点不重复的迹称为路。

（注：简单图中的路可以完全用顶点来表示，$P = v_{i_0}v_{i_1}\cdots v_{i_k}$）

闭途径(closed walk)：图G中起点和终点相同的途径称为闭途径。

闭迹(closed trail)：图G中边不重复的闭途径称为闭迹，也称为回路（circuit）。

圈(cycle)：中途点不重复的闭迹称为圈。

例 1.2 设G是一个简单图，若 $\delta(G) \geq 2$ ，则G中必含有圈。

证明：设G中的最长路为 $P = v_0\, v_1 \cdots v_k$ 。因 $d(v_0) \geq 2$ ，故存在与v_1相异的顶点v与v_0相邻。若$v \notin$ P ，则得到比P更长的路，这与P的取法矛盾。因此必定$v \in P$，从而G中有圈。证毕。

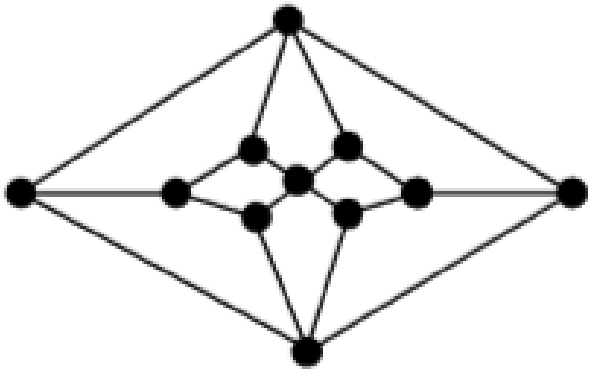
例 1.3 设G是简单图，若$\delta(G) \geq 3$, 则G必有偶圈。

6. 二部图

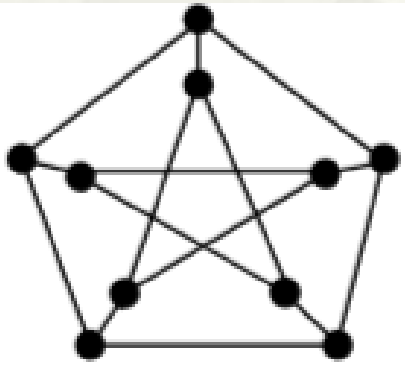
二部图 (bipartite graph)：若图G的顶点集可划分为两个非空子集X和Y，使得G的任一条边都有一个端点在X中，另一个端点在Y中，则称G为二部图，记为$G=(X\cup Y, E)$，(X, Y)称为G的一个顶点二划分。

完全二部图(complete bipartite graph)：在二部图$G=(X\cup Y, E)$中，若X的每个顶点与Y的每个顶点有边连接，则称G为完全二部图；若$|X|=m, |Y|=n$，则记此完全二部图为$K_{m,n}$。

例 1.4 判断下列图是不是二部图。



Herschel 图



Peterson 图

定理 1.1.2 一个图是二部图当且仅当它不含奇圈。

7. 连通性

图中两点的连通：如果在图G中u，v两点间有路相通，则称顶点u，v在图G中连通。

连通图（connected graph）：若图G中任意两个顶点都连通，则称图G是连通图。

图的连通分支（connected branch, component）：若图G的顶点集$V(G)$可划分为若干非空子集 $V_1, V_2, \cdots, V_w$ ，使得两顶点属于同一子集当且仅当它们在G中连通，则称每个子图$G[V_i]$为图G的一个连通分支$(i = 1,2,\cdots,w)$。

注:(1)图G的连通分支是G的一个极大连通子图。

(2)图G连通当且仅当ω=1。

例 1.5 设有2n个电话交换台，每个台与至少n个台有直通线路，则该交换系统中任二台均可实现通话。

证明：构造图G如下：以交换台作为顶点，两顶点间连边当且仅当对应的两台间有直通线路。问题化为：已知图G有2n个顶点，且$\delta(G) \geq n$，求证G连通。

事实上，假如G不连通，则至少有一个连通分支的顶点数不超过n。在此连通分支中，顶点的度至多是1-n。这与$\delta(G) \geq n$矛盾。证毕。

§1.2 最短路问题

一、赋权图

给图G的每条边e赋以一个实数$w(e)$，称为边e的**权**。每条边都赋有权的图称为**赋权图**。

权在不同的问题中会有不同的含义。例如交通网络中，权可能表示运费、里程或道路的造价等。

设H是赋权图G的一个子图，H的权定义为$W(H)=\sum_{e\in E(H)} w(e)$，特别地，对G中一条路P，其权为$W(P)=\sum_{e\in E(P)} w(e)$。

二、最短路问题

最短路问题：给定赋权图G及G中两点u，v，求u到v的具有最小权的路（称为u到v的最短路）。

注：赋权图中路的权也称为路的长，最短(u，v)路的长也称为u，v间的距离，记为d(u，v)。

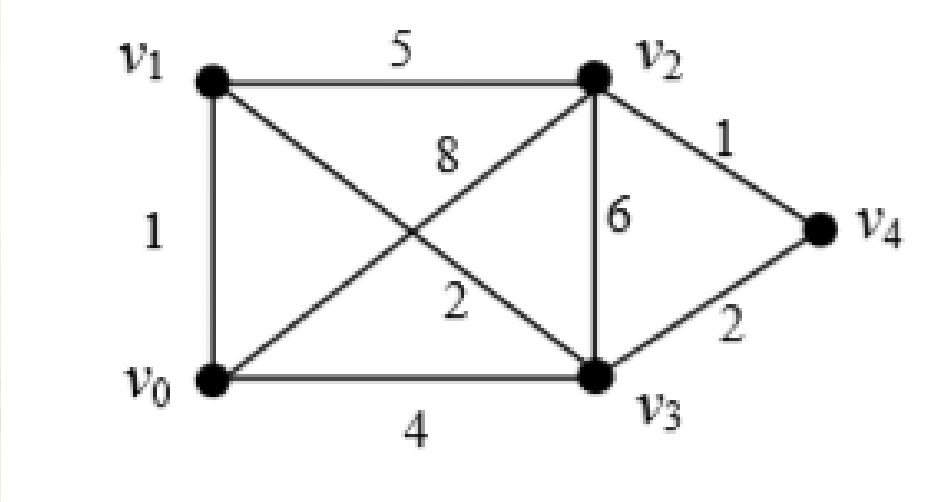
最短路问题是一个优化问题，属于网络优化和组合优化的范畴。对这种优化问题的解答一般是一个算法。最短路问题有很多算法，其中基本的一个是 Dijkstra 算法。

三、Dijkstra 算法

1. 算法思想

设赋权图G中所有边都具有非负权，Dijkstra算法的目标是求出G中某个指定顶点v_0到其它所有点的最短路。它依据的基本原理是：若路$P = v_0\ v_1 \cdots v_{k-1}\ v_k$ 是从v_0到v_k的最短路， 则$P' = v_0\ v_1 \cdots v_{k-1}$必是从$v_0$到$v_{k-1}$的最短路。基于这一原理，算法由近及远地逐次求出$v_0$到其它各点的最短路。

下面通过例子说明具体做法。



（1）令 $S=\{v_0\}, \bar{S}=V\backslash S$，求$v_0$到 $\bar{S}$ 中最近点的最短路。在当前的例子中，从v_0v_1、v_0v_2、 $v_0\ v_3$中选一条最短的，结果获得v_0到v_1的最短路v_0v_1。

（2）令$S := S \cup \{v_1\}$，　$\overline{S} := V \backslash S$，求$v_0$到$\overline{S}$中最近点的最短路。这里“最近”是$v_0$到$\overline{S}$的直接连边、以及从$v_0$ 出发的已有短路上的点（即S中除 v_0 外的点，当前只有 v_1 ）通过最短路再添加上向$\overline{S}$ 的连边所形成的路中最短的。即选取$\overline{S}$中一点v使得距离

$$d(v_0, v) = \min_{v_i \in S, v \in \overline{S}} \{d(v_0, v_i) + w(v_i v)\} \qquad (*)$$

在当前的例子中，从v_0v_2、v_0v_3、$v_0v_1v_2$、$v_0v_1v_3$中选一条最短的，结果获得v_0到v_3的最短路$v_0v_1v_3$。

(3) 令$S := S \cup \{v_3\}$，$\overline{S} := V \backslash S$，求$v_0$到$\overline{S}$中最近点的最短路。即选取$\overline{S}$中一点$v$使得

$$d(v_0, v) = \min_{v_i \in S, v \in \overline{S}} \{d(v_0, v_i) + w(v_i v)\}$$

当前应从v_0v_2、$v_0v_1v_2$、$v_0v_1v_3v_2$、$v_0v_1v_3v_4$中选一条最短的，结果获得v_0到v_4的最短路$v_0v_1v_3v_4$。

(4) 令$S := S \cup \{v_4\}$，$\overline{S} := V \backslash S$，求$v_0$到$\overline{S}$中最近点的最短路。即选取$\overline{S}$中一点$v$使得

$$d(v_0, v) = \min_{v_i \in S, v \in \overline{S}} \{d(v_0, v_i) + w(v_i v)\}$$

当前应从v_0v_2、$v_0v_1v_2$、$v_0v_1v_3v_2$、$v_0v_1v_3v_4v_2$中选一条最短的，结果获得v_0到v_2的最短路$v_0v_1v_2$(或$v_0v_1v_3v_4v_2$)。

一般地，若$S = \{v_0, v_1, \ldots, v_k\}$以及相应的最短路已找到，则可用(*)式来选取$v$，获得$v_0$到$v$的最短路。

Dijkstra 算法：求非负权图中 v_0点到其余各点的最短路。

第 1 步. 令 $l(v_0)=0$，$l(v)=\infty$，$(v\neq v_0)$，$S:=\{v_0\}$，$\overline{S}:=V\backslash S$，$i:=0$

第 2 步. 对每个$v\in\overline{S}$ 令 $l(v):=\min\{l(v),l(v_i)+w(v_iv)\}$。

取$v^*\in\overline{S}$使得$l(v^*)=\min\limits_{v\in\overline{S}}\{l(v)\}$。记$v_{i+1}=v^*$，令$S:=S\cup\{v_{i+1}\}$，

$\overline{S}:=V\backslash S$。

第 3 步.令$i:=i+1$。如果 $i=v-1$, 则停止；否则，转第 2 步。

3. 算法正确性

定理 1.2.1 Dijkstra 算法结束时，对任一个顶点 v，其标号 $l(v)$恰是 v_0 到 v 的最短路的长。

证明：按照算法，顶点 v 的最终标号 $l(v)$是 v 进入集合 S 时的标号。假设 v 在算法第 i 次循环时进入集合 S，则 $v = v_{i+1}$，且 v 进入集合 S 后 $S = \{v_0, v_1, \cdots, v_i, v_{i+1}\}$。由于 $l(v)$是算法在前 i 次循环中逐次比较出的当前从 v_0 到 v 最短路的长，因此从 v_0 到 v 仅含 $\{v_0, v_1, \cdots, v_i, v_{i+1}\}$ 中点的任何路的长都不会小于 $l(v)$。任取图 G 中一条从 v_0 到 v 的路 P，如果 P 仅含 $\{v_0, v_1, \cdots, v_i, v_{i+1}\}$ 中的点，则如上所述，P 的长不会小于 $l(v)$；如果 P 含 $\{v_0, v_1, \cdots, v_i, v_{i+1}\}$ 之外的点，无妨设沿着 P 从 v_0 出发第一个不在 $\{v_0, v_1, \cdots, v_i, v_{i+1}\}$ 中的顶点是 v'。因在算法第 i 次循环时，v_{i+1} 进入 S 而 v' 仍未进入 S，说明当前 $l(v_{i+1}) \leq l(v')$。注意 v' 只是 P 的中途点且图 G 中所有边的权都非负，而当前的 $l(v')$大于或等于 v_0 到 v' 仅含 $\{v_0, v_1, \cdots, v_i, v_{i+1}\}$ 中点的最短路的长，故路 P 的长大于或等于 P 上 v_0 到 v' 段的长（$\geq l(v')$），从而 P 的长不会小于 $l(v_{i+1})$（即 $l(v)$）。　证毕。

定理 1.2.2 Dijkstra 算法的计算复杂度为$O(v^2)$。

证明：Dijkstra 算法的主要计算量在第 2 步。在第i次循环中，第 2 步第 1 式需要$v-i-1$次加法，$v-i-1$次比较；第 2 式需要不超过$v-i-1$次比较$(i=0,1\cdots,v-2)$。从而$v-1$次循环总的计算量不超过

$$\sum_{i=0}^{v-2} 3(v-i-1) = \frac{3v(v-1)}{2} = O(v^2)$$

§1.3 树及其性质

不含圈的图称为森林(forest)，不含圈的连通图称为树（tree）。

定理 1.3.1 下列命题等价：

（1） G 是树；

（2） G 中无环边且任二顶点之间有且仅有一条路；

（3） G 中无圈且$\varepsilon = v - 1$；

（4） G 连通且$\varepsilon = v - 1$;

（5） G 连通且对任何$e \in E(G)$,$G - e$不连通；

（6） G 无圈且对任何$e \in E(\bar{G})$,$G + e$恰有一个圈。

引理1.3.1 若图 H 连通，则$\varepsilon(H) \geq v(H) - 1$。

推论 1.3.1 非平凡树至少含两个 1 度顶点（叶子）。

§1.4 生成树与最小生成树

一、生成树（spanning tree）

定义 1.4.1 设 T 是图 G 的一个子图，如果 T 是一棵树，且 $v(T) = v(G)$，则称 T 是 G 的一个生成树。

定理 1.4.1 每个连通图都有生成树。

证明：设 G 是一个连通图。令 $A = \{G'是G的连通子图且v(G') = v(G)\}$。易见 A 非空。从 A 中取边数最少的一个，记为 T。下证 T 是 G 的生成树。显然只需证明 T 是树即可。事实上，已知 T 连通，下证 T 无圈。若 T 有圈 C，则去掉 C 上任一条边 e，$T - e$仍连通。从而$T - e \subseteq A$。但$T - e$比 T 少一条边，这与 T 的取法矛盾。证毕。

二、最小生成树问题

最小生成树问题：在赋权图 G 中，求权最小的生成树（简称为最小生成树）。即：求 G 的一 棵生成树 T，使得

$$w(T) = \min_{e \in T} w(e)$$

（一） Kruskal 算法(Joseph Bernard Kruskal, 1956)

1.算法思想：先从图 G 中找出权最小的一条边作为最小生成树的边，然后逐次从剩余边中选边添入最小生成树中，每次选边找出不与已选边构成圈的权最小的一条边。直至选出 v (G)-1条边为止 。

2. 算法步骤：输入：赋权连通v阶图 G。输出：G 的最小生成树 T。

第一步，取 $e_1 \in E(G)$使得$w(e) = \min\limits_{e \in G}\{w(e)\}$，令 $i := 1$ 。

第二步，取$e_1 \in E(G)\backslash\{e_1, e_2, \dots, e_i\}$使得

(1) G[$\{e_1, e_2, \dots, e_i, e_{i+1}\}$] 不含圈； (2) e_{i+1}是满足(1)的权最小的边。

第三步，当 $i + 1 = v(G) - 1$ 时, 输出最小生成树 G[$\{e_1, e_2, \dots, e_{v-1}\}$]，算法停止；否则， 令 $i := i + 1$，转第二步。

3. 算法正确性：

定理1.4.2 设$e_1, e_2, \cdots, e_{v(G)-1}$是Kruskal算法获得的边，则边导出子图$G[\{e_1, e_2, \cdots, e_{v(G)-1}\}]$是 G 的最小生成树。

假设T^*不是权最小的生成树（下称最优树）。对G的任一棵不同于T^*的生成树T，记

$$f(T) = \min\{i | e_i \in \{e_1, e_2, \cdots, e_{v-1}\} \text{且} e_i \notin T\}$$

在 G 的所有最优树中选取一棵使$f(T)$最大的，记为$\tilde{T}$。$\tilde{T}$不会是T^*，因假设T^* 不是最优树)。设$f(\tilde{T}) = k$。由$f(\tilde{T})$的定义，$e_1, e_2, \cdots, e_{k-1}$既在$T^*$上也在$\tilde{T}$上，但$e_k$不在$\tilde{T}$上。因此$\tilde{T} + e_k$含有一个圈 C。C 上必有一条边$e_k' \notin T^*$。显然$T' = (\tilde{T} + e_k) - e_k'$也是一 棵生成树，且$w(T') = w(\tilde{T}) + w(e_k) - w(e_k')$。

按照算法，e_k 是使$G[\{e_1, e_2, \cdots, e_k\}]$中无圈的边中权最小的。注意$G[\{e_1, e_2, \cdots, e_{k-1}, e_k'\}]$是$\tilde{T}$的子图，也无圈。故由算法规则知：$w(e_k') \geq w(e_k)$。由前式，$w(T') \leq w(\tilde{T})$，这说明$T'$也是最优树。但$f(T') > k = f(\tilde{T})$（注意由于$e_1, e_2, \cdots, e_{k-1}$在$T^*$且$e_k' \notin T^*$，故$e_k' \notin \{e_1, e_2, \cdots, e_{k-1}\}$）。这与$\tilde{T}$的取法矛盾。证毕。

定理 1.4.3 若事先将边按权排序，则 Kruskal 算法的计算复杂度为$O(v^2)$；若加上事先排序的计算量，则 Kruskal 算法的计算复杂度为$O(v^2 \log_2 v)$。

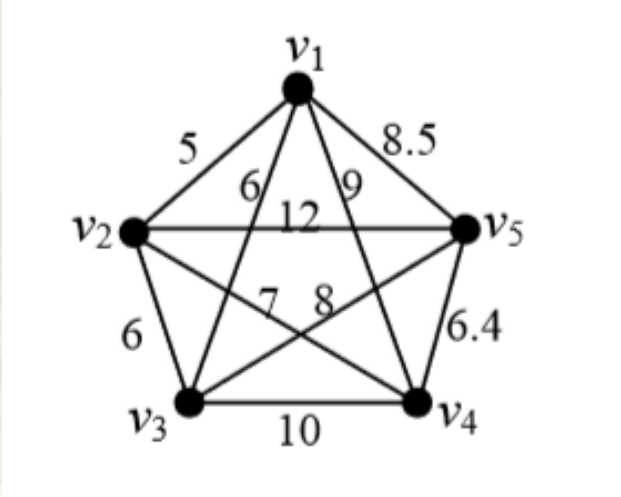
证明：算法执行过程中需要的主要计算量为

$$\varepsilon + v(v-1) \leq \frac{v(v-1)}{2} + v(v-1) = \frac{3}{2}v(v-1)$$

事先排序需要的计算量为$\varepsilon \log_2 \varepsilon \leq \frac{v(v-1)}{2} \log_2 \frac{v(v-1)}{2} \leq v^2 \log_2 v$。

故定理结论成立。

例 1.4.1 欲建设一个连接 5 个城市的光纤通信网络。各城市间线路的造价如下图所示，求一个使总造价最少的线路建设方案。



（二）Prim 算法(Robert Clay Prim, 1957)

1.算法思想: 先从图 G 中找出权最小的一条边作为最小生成树的边，在算法任一轮循环中， 设已经选出的边导出的子图为G′，从 G′的顶点向G′以外顶点的连边为E′，则选择E′中权最小的边向G′中添加，如此反复循环直至选出$v(G)-1$条边为止。

Prim 算法与 Kruskal 算法的根本区别在于：Kruskal 算法在保持无圈的基础上选边，而Prim 算法在保持连通的基础上选边。Prim 算法的添边过程实际上是树的生长过程。Kruskal 算法的添边过程一般情况下是森林合并为树的过程。

Kruskal 算法也称为避圈法，Prim 算法也称为边割法。

2. 算法步骤：

输入：赋权连通 v 阶图 G。输出：G 的最小生成树 T。

step1. 任取$v_0 \in V(G)$，令$S_0 = \{v_0\}$，$\overline{S_0} = V(G)\backslash S_0$，$i := 0$，$E_0 = \emptyset$。

step2. 求S_i到$\overline{S_i}$间权最小的边 e_{i+1}，设e_{i+1}的属于$\overline{S_i}$的端点为v_{i+1}，令$S_{i+1} := S_i \cup \{v_{i+1}\}$，$\overline{s}_{i+1} = V(G)\backslash S_{i+1}$，$E_{i+1} := E_i \cup \{e_{i+1}\}$

Step3. 当 $i+1 = v(G) - 1$时, 输出最小生成树 G[E_{i+1}]=G[$\{e_1, e_2, \ldots, e_{v-1}\}$]，算法停止；

否则，令$i := i + 1$，转 step2。

3. 算法正确性：

定理 1.4.4 设$e_1, e_2, \ldots, e_{v-1}$ 是 Prim 算法获得的边，则边导出子图 G[$\{e_1, e_2, \ldots, e_{v-1}\}$]是 G 的最小生成树。

4. 计算复杂度分析：

Prim 算法的主要计算量在第二步。在算法第 i 轮循环执行第二步时，S_i 中有 i 个顶点，$\overline{S_i}$中有$v-i$个顶点，故 S_i 到 $\overline{S_i}$间多有 $i\,(v-i)$ 条边。从这些边中选出一条权最小的边 e_{i+1}，需要 $i\,(v-i)-1$次比较。算法需循环执行第二步 $v-1$次，因此总的计算量为

$$\sum_{i=1}^{v-1}[i(v-i)-1] \leq \sum_{i=1}^{v-1} i(v-i) = \frac{1}{6}v(v-1)(v+1)。$$

由此可见，Prim 算法的计算复杂度为 $O(v^3)$。下面使用顶点标号法进一步降低 Prim 算法的计算复杂度。

（三）Prim 算法的标号形式—标号 Prim 算法

1.算法思想：给图的顶点赋以标号，该标号与边的权有关，在执行Prim算法的过程中，通过修改顶点标号和比较顶点的权，来选择满足 Prim 算法要求的小权边。

顶点 v 的标号记为 l(v)。用 w(uv)表示边e =uv 的权，若 u , v两点间没有边，则 w(uv)= ∞。

2. 算法步骤：输入：赋权连通 v 阶图 G。输出：G 的最小生成树 T。

step1. 任取$v_0 \in V(G)$，令$l(v_0) = 0$，$l(v) = \infty$，$(v \neq v_0)$，$S_0 = \{v_0\}$，$\overline{S_0} = V(G) \backslash S_0$，$T_0 = v_0$，$i := 0$；

step2. 对 $\forall v \in N_G(v_i) \cap \overline{S_i}$，若 $w(v_i v) < l(v)$，则令 $w(v_i v) = l(v)$ 。

step3. 选取 $v_{i+1} \in \overline{S_i}$使得$l(v_{i+1}) = \min\limits_{v \in S_i} l(v)$。设 $e_{i+1} = uv_{i+1}$，$(u \in S_i)$使得 $w(e_{i+1}) = l(v_{i+1})$，令 $S_{i+1} := S_i \cup = \{v_{i+1}\}$，$\overline{S_{i+1}} = V(G) \backslash S_{i+1}$，$T_{i+1} := T_i \cup \{e_{i+1}\}$。

step4. 当 $i + 1 = v(G) - 1$时, 输出小生成树 T_{v-1}，算法停止; 否则，令$i := i + 1$，转 step2。

3. 算法正确性：

标号 Prim 算法是 Prim 算法的标号实现，因此由Prim算法的正确性即知标号Prim算法结束时必能得到图 G的最小生成树。

4. 计算复杂度分析：

算法的主要计算量在第 2 步和第 3 步。算法第一次执行 Step3 需做 n−2 次比较，第二次执行 Step3 需做 n−3 次比较，……，因此执行 Step3 共需做 $\frac{1}{2}(v-2)(v-1)$次比较；同样，执行 Step2 共需不超过 $\frac{1}{2}(v-2)(v-1)$次比较。于是标号 Prim 算法的时间复杂性为 $O(v^2)$。

标号 Prim 算法将 Prim 算法每次循环中比较 S_{i+1}到 $\overline{S}_{i+1}$间所有边的权，改为修改 $N_G(e_i) \cap \overline{S_i}$中点的标号并比较 $\overline{S_i}$中点的标号，因此节省了计算量。