# 最小生成树算法及其应用

江苏省常州高级中学 吴景岳

【摘要】

最小生成树是图论中的经典问题,也是一个重要部分，一般书上往往只介绍求最小生成树的算法，而忽略了更精彩的算法应用部分。本文将对最小生成树算法及其应用作全面的分析说明，使大家对此有更加深刻的认识。本文分三部分：一、基础篇，主要介绍基础概念、求最小生成树的一般算法和常用算法。二、应用篇，具体问题具体分析，侧重于思考和证明的过程。三、总结。

【关键字】

最小生成树

【目录】

2、 应用篇

1．概述

2．例题

a) Robot——BOI2002

b) 北极通讯网络——Waterloo University 2002

3、 总结

【正文】

# 1．基础篇

## 1．1 定义

在电路设计中，常常需要把一些电子元件的插脚用电线连接起来。如果每根电线连接两个插脚，把所有 n 个插脚连接起来，只要用 n-1 根电线就可以了。在所有的连接方案中，我们通常对电线总长度最小的连接方案感兴趣。

把问题转化为图论模型就是：一个无向连通图 G=(V,E)，V 是插脚的集合，E 是插脚两两之间所有可能的连接的集合。给每条边(u,v)一个权值 w(u,v)，表示连接它们所需的电线长度。我们的目标就是找到一个无环的边集 T，连接其中所有的点且使总权值最小。

总权值 $w(T)=\sum_{(u,v)\in T} w(u,v)$

既然 T 是连接所有点的无环边集，它一定是一棵树。因为这棵树是从图 G 中生成出来的，我们把它叫做**生成树**。如果一棵生成树在所有生成树中总权值最小，我们就把它称作**最小生成树**。

## 1．2 求最小生成树的一般算法

解决最小生成树问题有没有一般的方法呢？下面我们就介绍一种贪心算法。该算法设置了集合 A，该集合一直是某最小生成树的子集。算法执行的每一步，都要决策是否把边(u,v)添加到集合 A 中，能够添加的条件是保证 A∪{(u,v)}仍然是最小生成树的子集。我们称像(u,v)这样的边为 A 的**安全边**，或称这样的边对集合 A 是安全的。

求最小生成树的一般算法流程如下：

```
GENERIC-MST(G,w)
1．A←Φ
2．while A没有形成一棵生成树
3．   do 找出A的一条安全边(u,v)
4．      A←A∪{(u,v)}
```

```
5. return A
```

一开始 A 为 Φ，显然满足最小生成树子集的条件。之后，每一次循环都把一条 A 的安全边加入 A 中，A 依然是最小生成树。

本节的余下部分将提出一条确认安全边的规则（定理 1），下一节将具体讨论运用这一规则寻找安全边的两个有效的算法。

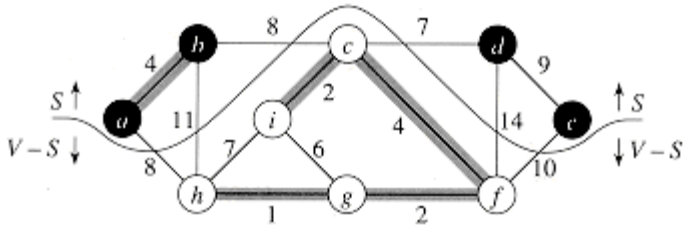



图 1 一个图的割(S,V−S)

首先定义几个概念。有向图 G=(V,E)的**割**(S,V-S)是 V 的一个分划。当一条边(u,v)∈E 的一个端点属于 S 而另一端点属于 V-S，我们说边(u,v)**通过**割(S,V-S)。若集合 A 中没有边通过割，就说割不**妨碍**集合 A。如果某边是通过割的具有最小权值的边，则称该边为通过割的一条**轻边**。

如图 1，集合 S 中的结点为黑色结点，集合 V-S 中的结点为白色结点。连接白色和黑色结点的那些边为通过该割的边。从边上所标的权值看，边(d,c)为通过该割的唯一一条轻边。子集 A 包含阴影覆盖的那些边，注意，由于 A 中没有边通过割，所以割(S,V-S)不妨碍 A。

确认安全边的规则由下列定理给出。

**定理 1 设图 G(V,E)是一无向连通图，且在 E 上定义了相应的实数值加权函数 w，设 A 是 E 的一个子集且包含于 G 的某个最小生成树中，割(S,V-S)是 G 的不妨碍 A 的任意割且边(u,v)是穿过割(S,V-S)的一条轻边，则边(u,v)对集合 A 是安全的。**

证明：设 T 是包含 A 的一最小生成树。如果 T 包含轻边（u,v），则 T 包含 A∪{(u,v)}，证明就完成了。否则，可设法建立另一棵包含 A∪{(u,v)}的最小生

成树 T’，(u,v)对集合 A 仍然是安全的。

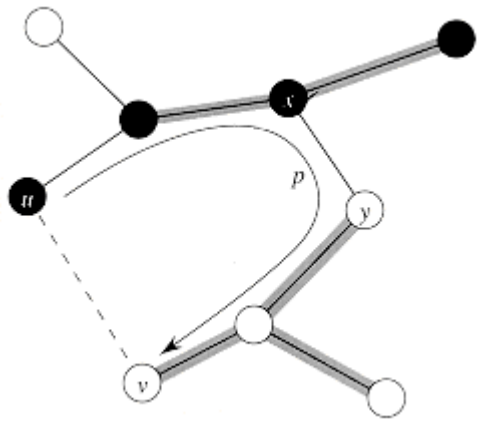



图 2 最小生成树 T’的建立

图 2 示出了最小生成树 T’的建立。S 中的结点为黑色，V-S 中的结点为白色，边(u,v)与 T 中从 u 到 v 的通路 P 构成一回路。由于边(u,v)通过割(S,V-S)，因此在 T 中的通路 P 上至少存在一条边也通过这个割。设(x,y)为满足此条件的边。因为割不妨碍 A，所以边(x,y)不属于 A。又因为(x,y)处于 T 中从 u 到 v 的唯一通路上，所以去掉边(x,y)就会把 T 分成两个子图。这时加入边(u,v)以形成一新的生成树 T’=(T-{(x,y)})∪{(u,v)}。

下一步我们证明 T’是一棵最小生成树。因为(u,v)是通过割(S,V-S)的一条轻边，且边(x,y)也通过割，所以有 w(u,v)≤w(x,y)，因此 w(T’)=w(T)-w(x,y)+w(u,v)≤w(T)。

但 T 是最小生成树，因此 T’必定也是最小生成树。又因为 T’包含 A∪{(u,v)}，所以(u,v)对 A 是安全的。（证毕）

通过定理 1 可以更好地了解算法 GENERIC-MST 在连通图 G=(V,E)上的执行流程。在算法执行过程中，集合 A 始终是无回路的，否则包含 A 的最小生成树就会出现一个环，这是不可能的。在算法执行中的任何一时刻，图 $G_A$=(V,A)

是一森林且$G_A$的每一连通支均为树。此外，对 A 安全的任何边(u,v)都连接$G_A$中不同的连通支，这是由于 A∪{(u,v)}必定不包含回路。

随着最小生成树的|V|-1 条边相继被确定，GENERIC-MST 中第 2-4 行的循环也随之要执行|V|-1 次。初始状态下，A=Φ，$G_A$中有|V|棵树，每个迭代过程均将减少一棵树，当森林中只包含一棵树时，算法执行终止。

下面再来看一个由定理 1 得出的推论，下一节中论述的两种算法均使用了这个推论。

**推论 1 设 G=(V,E)是一无向连通图，且在 E 上定义了相应的实数值加权函数 w，设 A 是 E 的子集且包含于 G 的某个生成树中，C 为森林$G_A$=(V,A)中的连通支。若边(u,v)是连接 C 和$G_A$中其它某连通支的一轻边，则边(u,v)对集合 A 是安全的。**

证明：因为割(C,V-C)不妨碍 A，因此(u,v)是该割的一条轻边。由定理 1，边(u,v)对集合 A 是安全的。（证毕）

## 1．3 常用算法

GENERIC-MST 是形成最小生成树的一般算法，不过我们需要的是更加具体的算法。这一节具体讨论两种常用的算法：Kruskal 算法和 Prim 算法。它们虽然都遵循所谓的“一般”算法，但在实现上有着各自的特点。

**Kruskal 算法：**

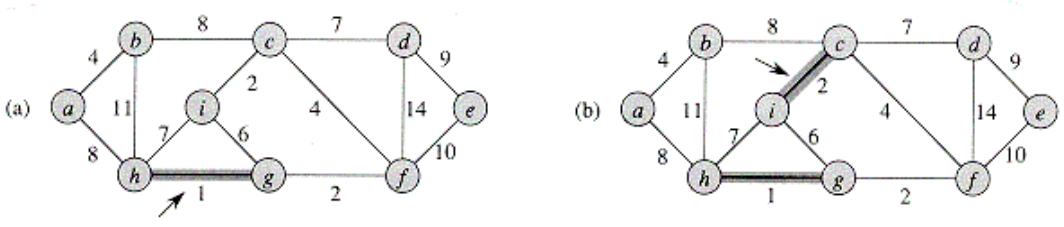
Kruskal 算法是直接建立在上一节中给出的一般最小生成树算法的基础之上的。该算法找出森林中连结任意两棵树的所有边中具有最小权值的边(u,v)作为安全边，并把它添加到正在生长的森林中。设 C1 和 C2 表示边(u,v)连结的两棵树。因为(u,v)必是连结 C1 和其它某棵树的一条轻边，所以由推论 1 可知(u,v)对 C1

是安全边。Kruskal 算法是一种贪心算法，因为算法每一步添加到森林中的边的权值都尽可能小。

Kruskal 算法的实现可以使用并查集。每一集合包含当前森林中某个树的结点，操作 FIND-SET(u)返回包含 u 的集合中的一个代表元素，因此我们可以通过测试 FIND-SET(u)是否等同于 FIND-SET(v)来确定两结点 u 和 v 是否属于同一棵树，通过过程 UNION 来完成树与树的连结。

```
MST-KRUSKAL(G,w)
1. A←Φ
2. for 每个结点 v∈V[G]
3.    do MAKE-SET(v)
4. 根据权 w 的非递减顺序对 E 的边进行排序
5. for 每条边(u,v)∈E，按权的非递减次序
6.    do if FIND-SET(u)≠FIND-SET(v)
7.          then A←A∪{(u,v)}
8.               UNION(u,v)
9. return A
```

Kruskal 算法的工作流程如图 3 所示。阴影覆盖的边属于正在生成的森林 A。算法按权的大小顺序考察各边。箭头指向算法每一步所考察到的边。第 1-3 行初始化集合 A 为空集并建立|V|棵树，每棵树包含图的一个结点。在第 4 行中根据其权值非递减次序对 E 的边进行排序。在第 5-8 行的 for 循环中首先检查对每条边(u,v)其端点 u 和 v 是否属于同一棵树。如果是，则把(u,v)加入森林就会形成一回路，所以这时放弃边(u,v)；如果不是，则两结点分属不同的树，由第 7 行把边加入集合 A 中，第 8 行对两颗树中的结点进行归并。



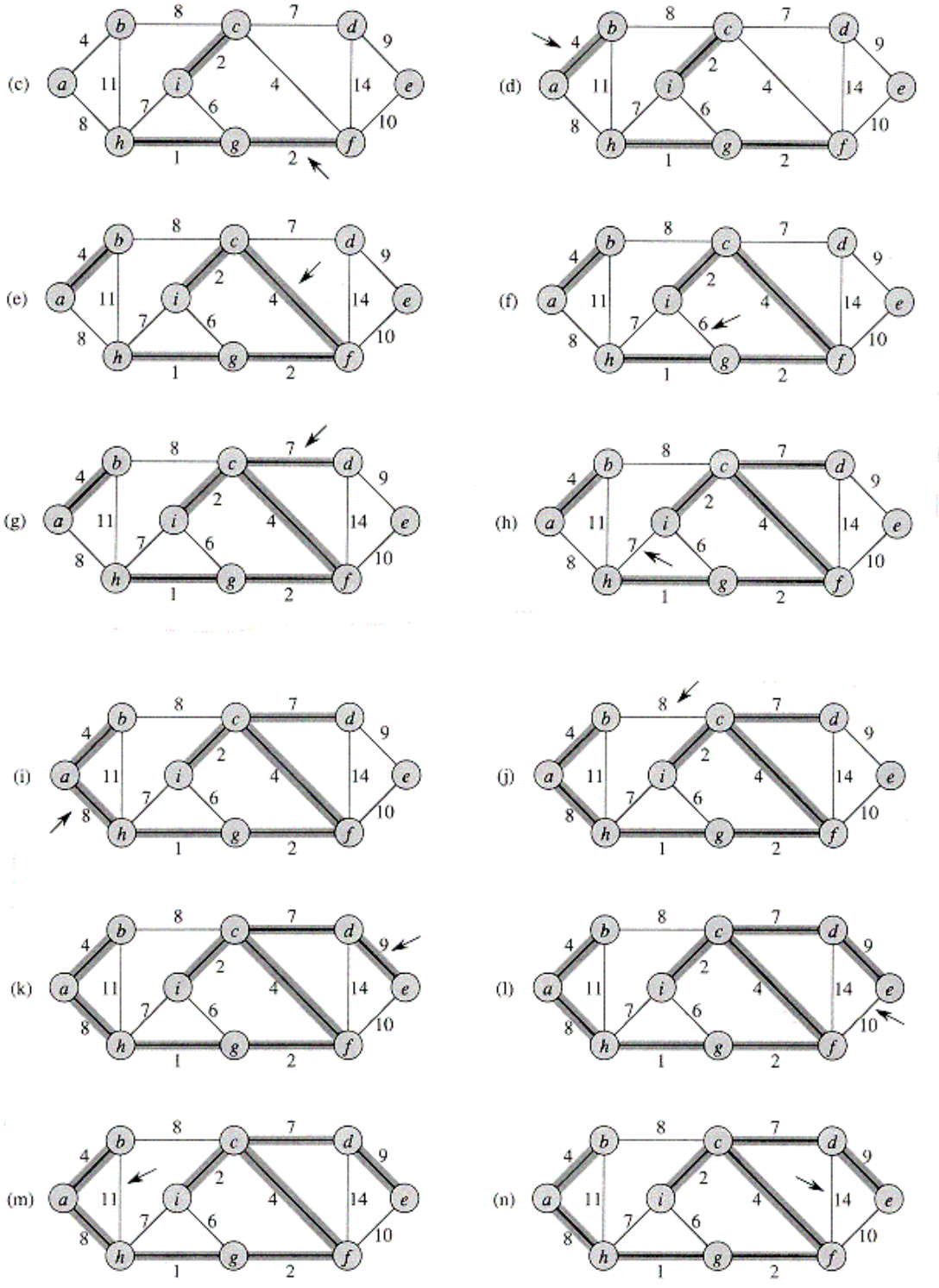



图 3 Kruskal 算法的执行流程

Kruskal 算法在图 G=(V,E)上的运行时间取决于如何实现集合的合并。我们可以采用路径压缩的优化方法，这是目前所知的最有效的。初始化需占用时间 O(V)，第 4 行中对边进行排序需要的运行时间为 O(ElgE)；对分离集的森林要进

行 O(E)次操作，总共需要时间为 O(Eα(E,V))，其中 α 函数为 Ackermann 函数的反函数。因为 α(E,V)=O(lgE)。所以 Kruskal 算法的全部运行时间为 O(ElgE)。

**Prim 算法：**

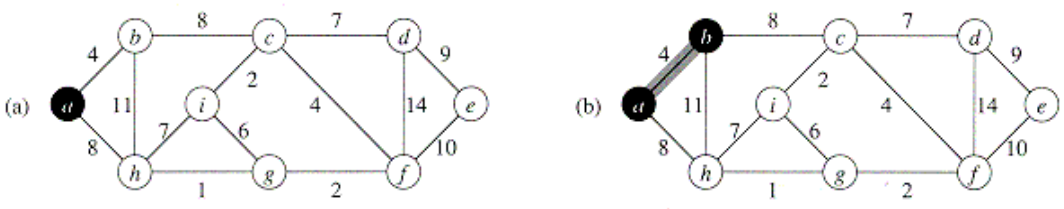
正如 Kruskal 算法一样。Prim 算法也是最小生成树一般算法的特例。它的执行非常类似于寻找图的最短通路的 Dijkstra 算法。Prim 算法的特点是集合 A 中的边总是只形成单棵树。如图 4 所示，阴影覆盖的边属于正在生成的树，树中的结点为黑色。在算法的每一步，树中与树外的结点确定了图的一个割，并且通过该割的轻边被加进树中。树从任意根结点 r 开始形成并逐渐生长直至该树跨越了 V 中的所有结点。在每一步，连接 A 中某结点到 V-A 中某结点的轻边被加入到树中。由推论 1，该规则总是加入对 A 安全的边，因此当算法终止时，A 中的边就成为一棵最小生成树。因为每次添加到树中的边都是使树的权尽可能小的边，因此上述策略是“贪心”的。

有效实现 Prim 算法的关键是设法较容易地选择一条新的边添加到由 A 的边所形成的树中，在下面的伪代码中，算法的输入是连通图 G 和将生成的最小生成树的根 r。在算法执行过程中，不在树中的所有结点都驻留于优先级基于 key 域的队列 Q 中，对每个结点 v，key[v]是连结 v 到树中结点的边所具有的最小权值；按常规，若不存在这样的边则 key[v]=∞。域 π[v]表示点 v 的“父亲结点”。在算法执行中，GENERIC-MST 的集合 A 隐含地满足：

$A=\{(v,\pi[v]):v \in V-\{r\}-Q\}$

当算法终止时，优先队列 Q 为空，因此 G 的最小生成树 A 满足：

$A=\{(v,\pi[v]): v \in V-\{r\}\}$



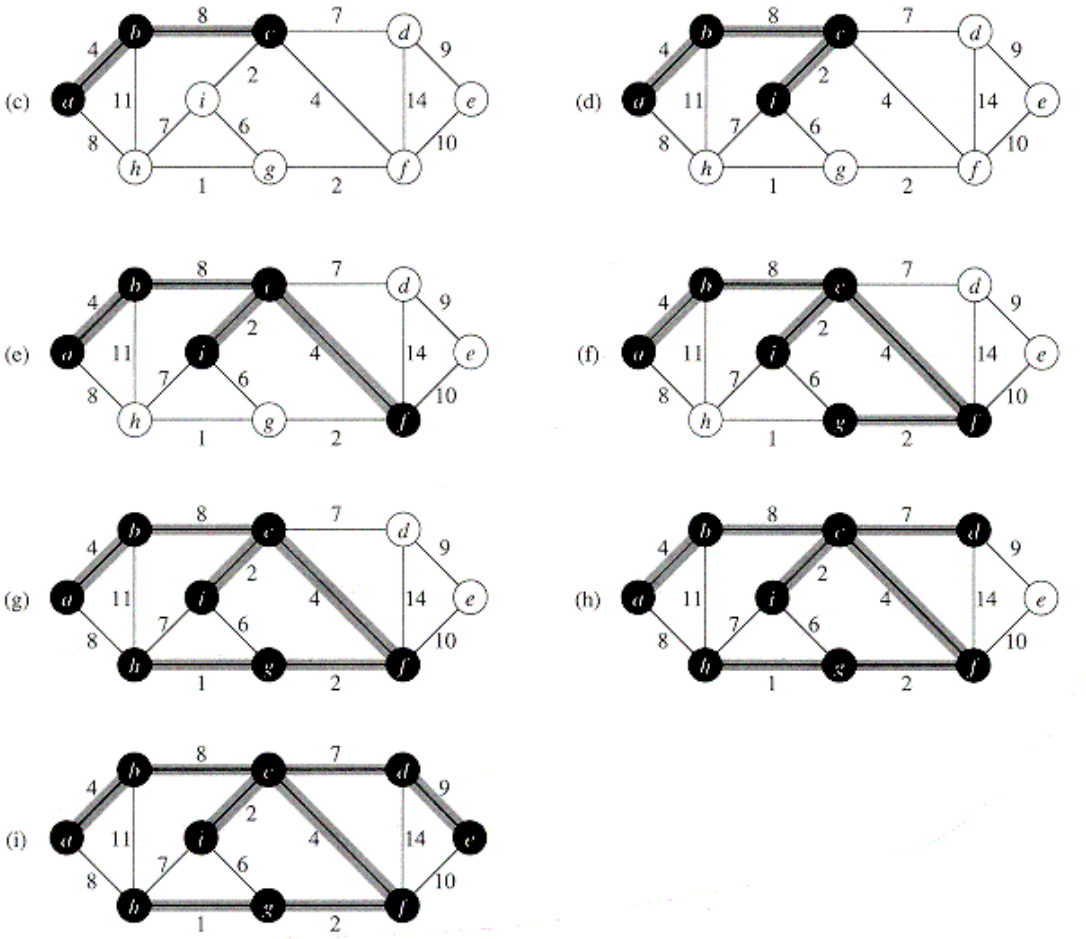



图 4 Prim 算法的执行流程

```
MST-PRIM(G,w,r)
1. Q←V[G]
2. for 每个u∈Q
3.   do key[u]←∞
4. key[r]←0
5. π[r]←NIL
6. while Q<>Φ
7.   do u←EXTRACT-MIN(Q) {返回队列Q中最小的元素}
8.      for 每个v∈Adj[u]
9.        do if v∈Q and w(u,v)<key[v]
10.            then π[v]←u
11.                 key[v]←w(u,v)
```

Prim 算法的工作流程如图 3 所示。第 1-4 行初始化优先队列 Q 使其包含所有结点，置每个结点的 key 域为∞（除根 r 以外），r 的 key 域被置为 0。第 5 行初始化 π[v]的值为 nil，这是由于 r 没有父亲结点。在整个算法中，集合 V-Q 包含正

在生长的树中的结点。第 7 行识别出与通过割(V-Q,Q)的一条轻边相关联的结点 u∈Q（第一次迭代例外，根据第 4 行这时 u=r）。从集合 Q 中去掉 u 后把它加入到树的结点集合 V-Q 中。第 8-11 行对与 u 邻接且不在树中的每个结点 v 的 key 域和 π 域进行更新，这样的更新保证 key[v]=w(v,π[v])且(v,π[v])是连结 v 和树中某结点的一条轻边。

Prim 算法的性能取决于我们如何实现优先队列 Q。若用二叉堆来实现 Q，第 1-4 行的初始化部分，其运行时间为 O(V)。第 6-11 行的循环需执行|V|次，且由于每次 EXTRACT-MIN 操作需要 O(lgV)的时间，所以对 EXTRACT-MIN 的总调用时间为 O(VlgV)。第 8-11 行的 for 循环总共要执行 O(E)次，这是因为所有邻接表的长度和为 2|E|。在 for 循环内部，第 9 行对队列 Q 的成员条件进行测试可以在常数时间内完成，这是由于我们可以为每个结点空出 1 位的空间来记录该结点是否在队列 Q 中，并在该结点被移出队列时随时对该位进行更新。第 11 行的赋值语句隐含一个对堆进行调整的操作，该操作在堆上可用 O(lgV)的时间完成。因此，Prim 算法的整个运行时间为 O(VlgV+ElgV)=O(ElgV)，和 Kruskal 算法的运行时间相同。

## 1．4 例题 1——Maintain（IOI2003）

**[题目大意]**

在一个初始化为空的无向图中，不断加入新边。如果当前图连通，就求出当前图最小生成树的总权值；否则，输出-1。

**[算法分析]**

根据数据规模，不可能每加入一条边都重新求一下最小生成树。所以我们的主要问题就是如何用较短的时间在一棵最小生成树中加入一条边并得到新的最小生成树。

生成树是具有 n 个点 n-1 条边的连通图，所以如果把一条新边加进去的话，就一定会形成一个环。然后再把环上的一条边删去（可以是新加进去的边），就

又形成了一棵生成树。为了保证是最小生成树，每次删除的边都必须是环上最长的。

于是不难得到下面的算法：

1. 初始化。
2. 读入新边，加入图中，直到连通或读完为止。
3. 如果不连通，转第 7 步。
4. 求出当前图的最小生成树。
5. 读入新边：x，y（端点），c（权值）。在旧图中找到从 x 到 y 的路径，如果路径上的最长边长于 c，则把新边加入，最长边去掉。
6. 未读完，转第 5 步；否则转第 7 步。
7. 结束。

**[小结]**

这道题目的难度并不大，其中蕴含了最小生成树的一些基本性质，比如：在最小生成树中加入一条新边，就会构成一个环；把这个环上的最长边去掉，又可以得到一棵新的最小生成树。

# 2．应用篇

和其它算法一样，要真正掌握最小生成树算法，不仅要理解算法本身的内容，还要学会如何在恰当的时候运用它。

下面我来看几个运用最小生成树解题的例子：

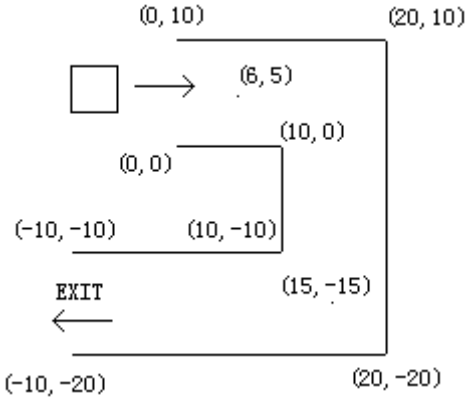
## 2．1 例题 2——Robot（BalkanOI2002）

**[问题描述]**

在不久的将来，机器人会在巴尔干信息学奥林匹克竞赛中把快餐传送给参赛者。他们将用一个简单的正方形盘子装所有的食物。不幸的是，厨房通往参赛者休息大厅的路上将会布满多种障碍物，因此，机器人不能搬运任意大小的盘子。你的任务是设计一个程序 ROBOT.EXE，用来确定提供食物的盘子的最大尺寸。

预想中的机器人将经过的路线应在由平行的墙构成的走廊中，而且走廊只能有 90º 拐角。走廊从 X 轴正方向开始。障碍物是柱子，由点表示，且都在走廊的两墙间。为了使机器人能够通过那条路，盘子不能碰到柱子或墙——只有盘子

的边缘能“靠”到他们。机器人和他的盘子只能在 X 轴或 Y 轴方向平移。假定机器人的尺寸小于盘子尺寸且机器人一直完全处于盘子下方。




**输入数据：**

第一行是一个整数 $m$ ($1 \le m \le 30$)，代表一堵墙有多少水平或竖直的段组成。在输入文件的下面 m+1 行是“上部”的墙的所有折点（包括端点）的坐标，也就是起始点有较大的坐标 y 的虚线。同样地，在下面 m+1 行是“下部”的墙的所有折点（包括端点）的坐标。输入文件下一行包含代表障碍物个数的整数 $n$ (0 $\le n \le$ 100)。在文件下面 n 行是障碍物坐标。所有坐标都是绝对值小于 32001 的整数。

**输出数据：**

只包含一个整数，表示满足题意的盘子最大边长。

**样例:**

**ROBOT.IN**

```
3
0 10
20 10
20 -25
-10 -25
0 0
10 0
10 -10
-10 -10
2
15 -15
6 5
```

**ROBOT.OUT**

5

**[题目大意]**

求能够通过一条有障碍走廊的正方形盘子的最大尺寸。

**[算法分析]**

一种比较容易想到的算法：二分+模拟(floodfill)。这种方法思路虽然简单，但编程复杂度很高。那么有没有更好的算法呢？

首先，换一个角度思考问题。根据题意机器人是正方形，障碍物是点。如果把机器人的尺寸移植到障碍物上，那么机器人就成了一个点，而障碍物就是正方形了。显然，这两个问题是等价的。要让一个点通不过，唯一的办法就是用障碍物把走廊堵住。这里的“堵住”就是说障碍物将在走廊的两堵墙之间形成一条通路。于是，这道题就被转化成了图论的问题。

把障碍物和墙壁看作图中的点，两点之间边的权值可以这样求得：定义两个点 p1(x1,y1)和 p2(x2,y2)的距离为 max{|x1-x2|,|y1-y2|}，障碍物的距离就是障碍点的距离；障碍物与墙壁的距离就是障碍点与墙壁上所有点的距离的最小值；两堵墙之间的距离就是走廊的最小宽度。

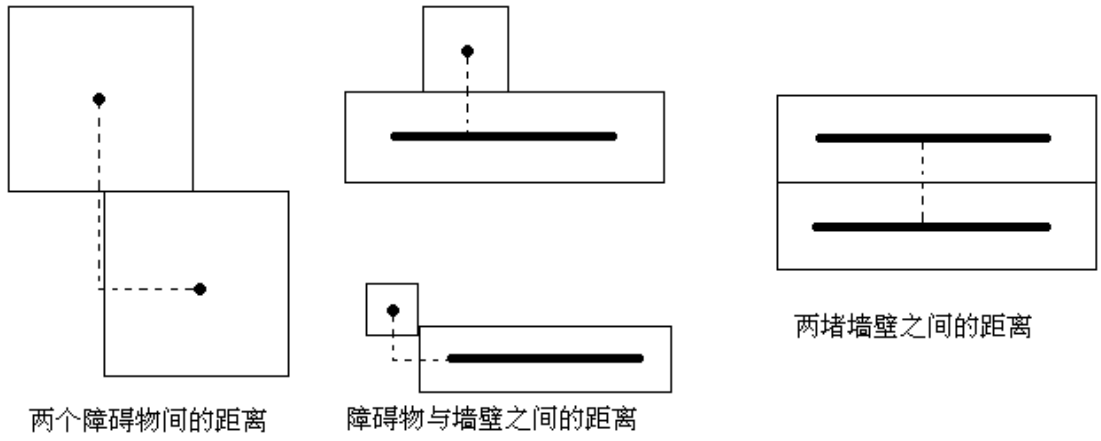



图 5 距离

把墙壁看作起点和终点，问题就转化为：从起点到终点的所有路径中最长边的最小值是多少？因为当边长达到一条路径上的最长边时，这条路径就“通”了，走廊也就被堵住了。

这个问题显然可以用二分+BFS 解决，时间复杂度是 $O(n^2 \log(\text{边的权值的范围}))$。不过我们有更好的算法：**先求出这个图的最小生成树，那么从起点到终点路径就是我们所要找的路径，这条路径上的最长边就是问题的答案。**这样时间复杂度就只有 $O(n^2)$了。可是怎么证明呢？

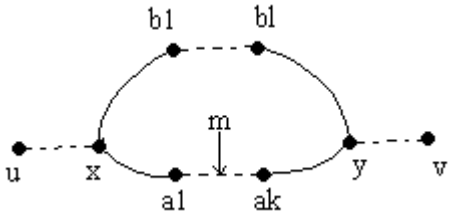
用反证法。

记起点为 u，终点为 v，最小生成树上从 u 到 v 的路径为旧路径，旧路径上

的最长边为 m。假设从 u 到 v 存在一条新路径且上面的最长边短于 m。

如果新路径包含 m，则新路径上的最长边不可能短于 m。与假设不符。

如果新路径不包含 m，则新路径一定“跨”过 m。如图：x-…-$a_1$-$a_2$-…$a_k$-y 是旧路径上的一段，x-$b_1$-$b_2$-…-$b_l$-y 是“跨”过去的一段。如果把 m 去掉，最小生成树将被分割成两棵子树，显然 x 和 y 分属于不同的子树（否则最小生成树包含一个环）。因此在 x-$b_1$-$b_2$-…-$b_l$-y 上，一定存在一条边 m’，它的端点分属于不同的子树。




因为最小生成树中只有 m 的端点分属于不同的子树，所以 m’不属于最小生成树。因此 m’和 m 一样是连接两棵子树的边。因为新路径的最长边短于 m 且 m’属于新路径，所以 w(m’)<w(m)。把 m 去掉，把 m’加入，将得到一棵新的生成树且它的总权值比最小生成树的还要小。显然不可能。

综上所述，不可能存在另一条从 u 到 v 的路径，使得它的最长边短于 m。最小生成树上从 u 到 v 的路径就是最长边最短的路径，该路径上的最长边就是问题的解。（证毕）

**[小结]**

此题有很多解法，除了刚才说的“二分+模拟”和“二分+BFS”之外，还可以用类似 Dijikstra 求最短路径的方法来做。既然如此，为什么偏要选择这种方法呢？不仅因为它效率高，也不仅因为它编程复杂度低，最重要的是因为它是最难想到的也是最有启发性的。就整个题目而言，把一道看似计算几何的题转化为图论模型就已经颇费脑筋了，再用最小生成树去解决这个图论题，真是难上加难。

在思考这道题时，因为问题是在从 u 到 v 的众多路径中选择最佳的，所以我在 u 和 v 之间画了两条路径，又标上了最长边，准备考虑如何比较它们的优劣。结果一个环出现了，而且环上还有一条最长边，于是眼前一亮：这不是最小生成树的基本图形吗？然后，经过不断的修正、繁琐的证明，这个算法才逐渐成熟起来。所以，最小生成树的应用并不是想象的那么简单，它需要敏锐的洞察力、

扎实的图论基础和严谨的思维。

## 2．2 例题 3——北极通讯网络（Waterloo University 2002）

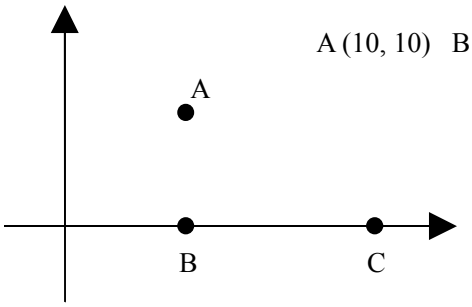
**[问题描述]**

北极的某区域共有 n 座村庄( $1 \le n \le 500$ )，每座村庄的坐标用一对整数(x, y)表示，其中 $0 \le x, y \le 10000$。为了加强联系，决定在村庄之间建立通讯网络。通讯工具可以是无线电收发机，也可以是卫星设备。所有的村庄都可以拥有一部无线电收发机， 且所有的无线电收发机型号相同。但卫星设备数量有限，只能给一部分村庄配备卫星设备。

不同型号的无线电收发机有一个不同的参数 d，两座村庄之间的距离如果不超过 d 就可以用该型号的无线电收发机直接通讯，d 值越大的型号价格越贵。拥有卫星设备的两座村庄无论相距多远都可以直接通讯。

现在有 k 台（$0 \le k \le 100$）卫星设备，请你编一个程序，计算出应该如何分配这 k 台卫星设备，才能使所拥有的无线电收发机的 d 值最小，并保证每两座村庄之间都可以直接或间接地通讯。

例如，对于下面三座村庄：




其中 $|AB|=10 \quad |BC|=20 \quad |AC|=10\sqrt{5} \approx 22.36$

如果没有任何卫星设备或只有 1 台卫星设备（k=0 或 k=1），则满足条件的最小的 d = 20，因为 A 和 B，B 和 C 可以用无线电直接通讯；而 A 和 C 可以用 B 中转实现间接通讯（即消息从 A 传到 B，再从 B 传到 C）；

如果有 2 台卫星设备（k=2），则可以把这两台设备分别分配给 B 和 C，这

样最小的 d 可取 10，因为 A 和 B 之间可以用无线电直接通讯；B 和 C 之间可以用卫星直接通讯；A 和 C 可以用 B 中转实现间接通讯。

如果有 3 台卫星设备，则 A,B,C 两两之间都可以直接用卫星通讯，最小的 d 可取 0。

**[算法分析]**

当正向思考受阻时，逆向思维可能有奇效。本题就是这样。知道卫星设备的数量，求最小的收发距离，可能比较困难；如果知道距离求数量，就很简单了。把所有可以互相通讯的村庄连接起来，构成一个图。卫星设备的台数就是图的连通支的个数。

问题转化为：找到一个最小的 d，使得把所有权值大于 d 的边去掉之后，连通支的个数小于等于 k。

先看一个定理。**定理 2：如果去掉所有权值大于 d 的边后，最小生成树被分割成为 k 个连通支，图也被分割成为 k 个连通支。**

证明：

用反证法。假设原图被分割成 k’ (k'≠k)个连通支，显然不可能 k’>k，所以 k’<k。因此在某一图的连通支中，最小生成树被分成了至少两部分，不妨设其为 T1,T2 。因为 T1 和 T2 同属于一个连通支，所以一定存在 x∈T1，y∈T2，w(x,y)≤d。又因为在整个最小生成树中，所以 x 到 y 的路径中一定存在一条权值大于 d 的边(u,v)（否则 x 和 y 就不会分属于 T1 和 T2 了），w(x,y)≤d<w(u, v)，所以把(x,y)加入，把(u,v)去掉，将得到一棵总权值比最小生成树还小的生成树。这显然是不可能的。所以，原命题成立。（证毕）

有了这个定理，很容易得到一个构造算法：**最小生成树的第 k 长边就是问题的解。**

证明：

首先，d 取最小生成树中第 k 长的边是可行的。如果 d 取第 k 长的边，我们

将去掉最小生成树中前 k-1 长的边，最小生成树将被分割成为 k 部分。由定理 2，原图也将分割成为 k 部分。（可行性）

其次，如果 d 比最小生成树中第 k 长的边小的话，最小生成树至少被分割成为 k+1 部分，原图也至少被分割成为 k+1 部分。与题意不符。（最优性）

综上所述，最小生成树中第 k 长的边是使得连通支个数≤k 的最小的 d，即问题的解。（证毕）

**[小结]**

一道看似毫无头绪的题目被我们用简单的武器拿下了，这就是最小生成树的威力。就此题而言，定理 2 是相当重要的，它在最小生成树和图的连通支之间搭起了一座桥梁，正是这座桥梁使我们顺利地到达了胜利的彼岸。

## 3．总结

本文所介绍的最小生成树算法的应用只能算是“冰山一角”，围绕最小生成树的定理和应用还有很多，远远不是一篇论文所能涵盖的。这些定理和应用有的已经被人们发现，而更多更精彩的还等待着后来人的挖掘。

最小生成树是图论中相当基础的算法之一，可是细细挖掘之后，得到的是惊人的收获。其实，不光是最小生成树，还有相当一部分基础的算法或者定理，在很简单很显然的表面背后，隐藏着令人回味的精髓。这些精髓正是需要发现的。

说到发现，我想起了下面四句话。虽然科学发现多种多样，但这四句话是无数科学发现者们共同的经历。

**博览参考文献。**“站在巨人的肩膀上”，才能看得更远。

**磨砺智慧之剑。**智慧是战胜困难的武器。

**认定大致方向。**认准一条路，坚定不移的走下去。

**小心探索前进。**同时不放过任何有价值的线索，它们或许正是问题的突破口。

但愿这四句话能给大家带来些有益的启示。

## 【感谢】

南京金陵中学的苏展同学帮我证明了“北极通讯网络”中的定理 2，在此表示感谢！

## 【参考资料】


| 书名 | 出版社 | 编者 |
| --- | --- | --- |
| 《Introduction to Algorithms》（Second Edition） | The MIT Press | Thomas H.Cormen<br>Charles E.Leiserson<br>Ronald L.Rivest<br>Clifford Stein |
| 《走向数学发现》 | 大象出版社 | 蒋声 |

附录一：Prim 算法（用二叉堆实现）

```
Program MST_Prim;

Const
  inputfile='mst.in';
  outputfile='mst.out';
  maxn=200;

Type
  link=^node;{邻接表类型}
  node=Record
    data,cost:longint;
    next:link;
  End;
  rtype=Record{二叉堆元素类型}
    point,cost:longint;
  End;

Var
  a:Array [1..maxn] of link;{邻接表}
  h:Array [1..maxn] of rtype;{堆}
  pos:Array [1..maxn] of longint;{每个点在堆中的位置}
  n,m,htail,ans:longint;

  Procedure Insert(x,y,c:longint);{把边插入邻接表中}
  Var
    d:link;
  Begin
    New(d); d^.data:=y; d^.cost:=c;
    d^.next:=a[x]; a[x]:=d;
  End;

  Procedure Init;{初始化}
  Var
    i,x,y,c:longint;
  Begin
    Assign(input,inputfile);
    Reset(input);
    Read(n,m);
    For i:=1 to m Do
    Begin
      Read(x,y,c);
      Insert(x,y,c); Insert(y,x,c);
    End;
```

```
  Close(input);
End;

Procedure Go_Up(j:longint);{堆中元素向上调整}
Var
  x:rtype;
  i:longint;
Begin
  x:=h[j];
  While j>1 Do
  Begin
    i:=j div 2;
    If h[i].cost<=x.cost Then Break;
    h[j]:=h[i]; pos[h[j].point]:=j;
    j:=i;
  End;
  h[j]:=x; pos[x.point]:=j;
End;

Procedure Go_Down(i:longint);{堆中元素向下调整}
Var
  x:rtype;
  j:longint;
Begin
  x:=h[i];
  While i+i<=htail Do
  Begin
    j:=i+i;
    If (j<htail) and (h[j+1].cost<h[j].cost) Then Inc(j);
    If x.cost<=h[j].cost Then Break;
    h[i]:=h[j]; pos[h[i].point]:=i;
    i:=j;
  End;
  h[i]:=x; pos[x.point]:=i;
End;

Procedure Pop(Var point,cost:longint);{最小的元素出堆}
begin
  point:=h[1].point; cost:=h[1].cost;
  pos[point]:=-1;
  h[1]:=h[htail]; pos[h[1].point]:=1;
  Dec(htail);
  Go_Down(1);
End;
```

```
  Procedure Main;{主过程}
  Var
    i,pp,c,j:longint;
    p:link;
  Begin
    For i:=1 to n Do{堆的初始化}
    Begin
      h[i].point:=i;
      If i=1 Then h[i].cost:=0 Else h[i].cost:=maxlongint;
      pos[i]:=i;
    End;
    htail:=n;
    ans:=0;
    For pp:=1 to n Do
    Begin
      Pop(i,c); Inc(ans,c);{最小的元素出堆}
      p:=a[i];{对其它元素的值进行修改}
      While p<>nil Do
      Begin
        j:=p^.data;
        If (pos[j]<>-1) and (p^.cost<h[pos[j]].cost) Then
        Begin
          h[pos[j]].cost:=p^.cost;
          Go_Up(pos[j]);
        End;
        p:=p^.next;
      End;
    End;
  End;

  Procedure Answer;{输出}
  Begin
    Assign(output,outputfile);
    Rewrite(output);
    Writeln(ans);
    Close(output);
  End;

Begin
  Init;
  Main;
  Answer;
End.
```

附录二：Kruskal 算法

```
Program MST_Kruskal;

Const
  inputfile='mst.in';
  outputfile='mst.out';
  maxn=200;
  maxe=maxn*maxn;

Type
  edgetype=Record{边的类型}
    x,y,c:longint;
  End;

Var
  e:Array [1..maxe] of edgetype;{边集}
  f:Array [1..maxn] of longint;{并查集}
  n,m,ans:longint;

  Procedure Init;{初始化}
  Var
    i:longint;
  Begin
    Assign(input,inputfile);
    Reset(input);
    Read(n,m);
    For i:=1 to m Do Read(e[i].x,e[i].y,e[i].c);
    Close(input);
  End;

  Procedure Qksort(p,q:longint);{快速排序}
  Var
    r,i,j:longint;
    k:edgetype;
  Begin
    r:=Random(q-p+1)+p;
    k:=e[r];
    e[r]:=e[p];
    i:=p; j:=q;
    While i<j Do
    Begin
      While (i<j) and (k.c<=e[j].c) Do Dec(j);
      e[i]:=e[j];
```

```
    While (i<j) and (e[i].c<=k.c) Do Inc(i);
    e[j]:=e[i];
  End;
  e[i]:=k;
  If p<i-1 Then Qksort(p,i-1);
  If i+1<q Then Qksort(i+1,q);
End;

Function top(i:longint):longint;{并查集中点i所在树的根}
Begin
  If i<>f[i] Then f[i]:=top(f[i]);
  top:=f[i];
End;

Procedure Union(i,j,c:longint);{合并i和j所在集合}
Var
  x,y:longint;
Begin
  x:=top(i); y:=top(j);
  If x<>y Then
  Begin
    Inc(ans,c);
    f[y]:=x;
  End;
End;

Procedure Main;{主过程}
Var
  i:longint;
Begin
  Qksort(1,m);
  For i:=1 to n Do f[i]:=i;
  ans:=0;
  For i:=1 to m Do Union(e[i].x,e[i].y,e[i].c);
End;

Procedure Answer;{输出}
Begin
  Assign(output,outputfile);
  Rewrite(output);
  Writeln(ans);
  Close(output);
End;
```

```
Begin
  Init;
  Main;
  Answer;
End.
```

附录三：Maintain 源程序

```
Program Maintenance;

Const
  maxn=200;

Type
  rtype=Record{Prim 算法中队列元素的类型}
    point,cost:longint;
  End;

Var
  a:Array [1..maxn,1..maxn] of longint;{用邻接矩阵表示边集}
  dep,f:array [1..maxn] of longint;{每个点在最小生成树中的深度即父亲结点}
  edge,n,m,i,j,cost,k,max,mi,mj,ans:longint;

  Function Connected:boolean;{判断当前图是否连通}
  Var
    q:Array [1..maxn] of longint;
    visit:Array [1..maxn] of boolean;
    head,tail,i,j:longint;
  Begin
    Fillchar(visit,Sizeof(visit),0);
    head:=1; tail:=1; q[1]:=1; visit[1]:=true;
    While head<=tail Do
    Begin
      i:=q[head];
      For j:=1 to n Do
        If not(visit[j]) and (a[i,j]>0) Then
        Begin
          visit[j]:=true;
          Inc(tail); q[tail]:=j;
        End;
      Inc(head);
    End;
    Connected:=(tail=n);
  End;
```

```
Procedure MST;{用Prim算法求最小生成树}
Var
  b:Array [1..maxn] of rtype;
  i,j,temp,tot:longint;
  tb:rtype;
Begin
  Fillchar(f,Sizeof(f),0);
  For i:=1 to n Do
  Begin
    b[i].point:=i;
    If i=1 Then b[i].cost:=0 Else b[i].cost:=maxlongint;
  End;
  For i:=1 to n-1 Do
  Begin
    For j:=i+1 to n Do
      If b[j].cost<b[i].cost Then
      Begin
        tb:=b[i]; b[i]:=b[j]; b[j]:=tb;
      End;
    For j:=i+1 to n Do
    Begin
      temp:=a[b[i].point,b[j].point];
      If (temp>0) and (temp<b[j].cost) Then
      Begin
        b[j].cost:=temp;
        f[b[j].point]:=b[i].point;
      End;
    End;
  End;
  Fillchar(a,Sizeof(a),0);
  tot:=0;
  For i:=2 to n Do
  Begin
    j:=b[i].point; Inc(tot,b[i].cost);
    a[j,f[j]]:=b[i].cost; a[f[j],j]:=b[i].cost;
  End;
  Writeln(tot);
End;

Function ComputDep(i:longint):longint;
{计算每个点的深度，此过程用记忆化搜索}
Begin
  If dep[i]=-1 Then dep[i]:=ComputDep(f[i])+1;
  ComputDep:=dep[i];
```

```
End;

Procedure GetMax(i,j:longint; Var max,mi,mj:longint);
{得到最小生成树上从I到j路径上的最长边}
Var
  backupi,backupj,k:longint;
Begin
  max:=0;
  backupi:=i; backupj:=j;
  While dep[i]>dep[j] Do i:=f[i];
  While dep[i]<dep[j] Do j:=f[j];
  While i<>j Do Begin i:=f[i]; j:=f[j] End;
  k:=i;{k是I和j的深度最大的共同祖先}
  i:=backupi; j:=backupj;
  While i<>k Do
  Begin
    If a[i,f[i]]>max Then
    Begin
      max:=a[i,f[i]];
      mi:=f[i]; mj:=i;
    End;
    i:=f[i];
  End;
  While j<>k Do
  Begin
    If a[j,f[j]]>max Then
    Begin
      max:=a[j,f[j]];
      mi:=f[j]; mj:=j;
    End;
    j:=f[j];
  End;
End;

Procedure DFS(i:longint);{用深度优先得到一棵搜索树，也即最小生成树}
Var
  j:longint;
Begin
  For j:=1 to n Do
    If (j<>f[i]) and (a[i,j]>0) Then
    Begin
      f[j]:=i;
      DFS(j);
    End;
```

```
  End;

Begin
  Assign(input,'maintain.in');
  Assign(output,'maintain.out');
  Reset(input);
  Rewrite(output);

  Read(n,m);
  Fillchar(a,Sizeof(a),0);
  For edge:=1 to m Do{不断读入新边，直到连通为止}
  Begin
    Read(i,j,a[i,j]); a[j,i]:=a[i,j];
    If Connected
      Then Begin
        MST;
        Break;
      End
      Else Writeln(-1);
  End;

  While edge<m Do
  Begin
    Inc(edge);
    Read(i,j,cost);
    dep[1]:=0; For k:=2 to n Do dep[k]:=-1;{用记忆化搜索求每个点的深度}
    For k:=2 to n Do dep[k]:=ComputDep(k);
    GetMax(i,j,max,mi,mj);
    If max>cost Then{如果I到j路径上的最长边大于新边长度，立即更新}
    Begin
      a[mi,mj]:=0; a[mj,mi]:=0;
      a[i,j]:=cost; a[j,i]:=cost;
      Fillchar(f,Sizeof(f),0);
      DFS(1);
    End;
    ans:=0;{得到最小生成树的总权值}
    For k:=2 to n Do Inc(ans,a[k,f[k]]);
    Writeln(ans);
  End;

  Close(output);
  Close(input);
End.
```

附录四：Robot 源程序

```
program Robot;

const
  maxm=30;
  maxn=100;

type
  ptype=record{点的类型}
    x,y:longint;
  end;
  rtype=record{队列中元素的类型}
    point,cost:longint;
  end;

var
  w1,w2:array [1..maxm+1] of ptype;{走廊的墙壁}
  p:array [1..maxn] of ptype;{障碍物}
  a:array [1..maxn+2,1..maxn+2] of longint;{用邻接矩阵存放所有的边}
  b:array [1..maxn+2] of rtype;{队列}
  f:array [1..maxn+2] of longint;{最小生成树中每个点的父亲结点}
  m,i,n,j,temp,k,ans:longint;
  tb:rtype;

  function min(a,b:longint):longint;
  begin
    if a<b then min:=a else min:=b;
  end;

  function max(a,b:longint):longint;
  begin
    if a>b then max:=a else max:=b;
  end;

begin
  assign(input,'robot.in');{文件读入}
  assign(output,'robot.out');
  reset(input);
  read(m);
  for i:=1 to m+1 do read(w1[i].x,w1[i].y);
  for i:=1 to m+1 do read(w2[i].x,w2[i].y);
  read(n);
  for i:=1 to n do read(p[i].x,p[i].y);
  close(input);
```

```
{建边的过程，程序中点1~n表示障碍物，n+1和n+2表示走廊的墙壁}
  for i:=1 to n+2 do
    for j:=1 to n+2 do
      a[i,j]:=maxlongint;
  for j:=1 to n do
    for i:=1 to m do
    begin
      if w1[i].x=w1[i+1].x
        then
          if (p[j].y-w1[i].y)*(p[j].y-w1[i+1].y)<=0
            then temp:=abs(w1[i].x-p[j].x)
                 else  temp:=min(max(abs(p[j].x-w1[i].x),abs(p[j].y-
w1[i].y)),max(abs(p[j].x-w1[i+1].x),abs(p[j].y-w1[i+1].y)))
        else
          if (p[j].x-w1[i].x)*(p[j].x-w1[i+1].x)<=0
            then temp:=abs(w1[i].y-p[j].y)
                 else  temp:=min(max(abs(p[j].x-w1[i].x),abs(p[j].y-
w1[i].y)),max(abs(p[j].x-w1[i+1].x),abs(p[j].y-w1[i+1].y)));
      if temp<a[n+1,j] then
      begin
        a[n+1,j]:=temp;
        a[j,n+1]:=temp;
      end;
    end;
  for j:=1 to n do
    for i:=1 to m do
    begin
      if w2[i].x=w2[i+1].x
        then
          if (p[j].y-w2[i].y)*(p[j].y-w2[i+1].y)<=0
            then temp:=abs(w2[i].x-p[j].x)
                 else  temp:=min(max(abs(p[j].x-w2[i].x),abs(p[j].y-
w2[i].y)),max(abs(p[j].x-w2[i+1].x),abs(p[j].y-w2[i+1].y)))
        else
          if (p[j].x-w2[i].x)*(p[j].x-w2[i+1].x)<=0
            then temp:=abs(w2[i].y-p[j].y)
                 else  temp:=min(max(abs(p[j].x-w2[i].x),abs(p[j].y-
w2[i].y)),max(abs(p[j].x-w2[i+1].x),abs(p[j].y-w2[i+1].y)));
      if temp<a[j,n+2] then
      begin
        a[j,n+2]:=temp;
        a[n+2,j]:=temp;
      end;
```

```
    end;
  for i:=1 to n do
    for j:=1 to n do
      a[i,j]:=max(abs(p[i].x-p[j].x),abs(p[i].y-p[j].y));

  fillchar(f,sizeof(f),0);{用 Prim 算法求最小生成树}
  for i:=1 to n+2 do
  begin
    b[i].point:=i;
    if i=n+1 then b[i].cost:=0 else b[i].cost:=maxlongint;
  end;
  for i:=1 to n+1 do
  begin
    for j:=i+1 to n+2 do
      if b[j].cost<b[i].cost then
      begin
        tb:=b[i]; b[i]:=b[j]; b[j]:=tb;
      end;
    for j:=i+1 to n+2 do
    begin
      temp:=a[b[i].point,b[j].point];
      if temp<b[j].cost then
      begin
        b[j].cost:=temp;
        f[b[j].point]:=b[i].point;
      end;
    end;
  end;

  ans:=0;{从 n+1 到 n+2 路径上的最长边}
  i:=n+2;
  while i<>n+1 do
  begin
    if a[i,f[i]]>ans then ans:=a[i,f[i]];
    i:=f[i];
  end;

  rewrite(output);
  writeln(ans);
  close(output);
end.
```

附录五：“北极通讯网络”源程序

```
Program Network;
```

```
Const
  inputfile='input.txt';
  outputfile='output.txt';
  maxn=500;
  maxnum=1e+20;

Type
  ptype=Record{平面上点的类型}
    x,y:longint;
  End;
  rtype=Record{队列元素的类型}
    point:longint;
    cost:real;
  End;

Var
  a:Array [1..maxn,1..maxn] of real;{两点间的距离}
  p:Array [1..maxn] of ptype;{平面上所有的点}
  f:Array [1..maxn] of longint;{最小生成树中每个点的父亲结点}
  b:Array [1..maxn] of rtype;{队列}
  e:Array [1..maxn] of real;{最小生成树中的所有边}
  n,k:longint;

  Function Dis(x1,y1,x2,y2:longint):real;{两点间的距离}
  Begin
    Dis:=Sqrt(Sqr(x1-x2)+Sqr(y1-y2));
  End;

  Procedure Init;{初始化}
  Var
    i,j:longint;
  Begin
    Assign(input,inputfile);
    Reset(input);
    Read(n,k); If k<1 Then k:=1; If k>n Then k:=n;
    For i:=1 to n Do Read(p[i].x,p[i].y);
    Close(input);
    For i:=1 to n Do
      For j:=1 to n Do
        a[i,j]:=Dis(p[i].x,p[i].y,p[j].x,p[j].y);
  End;

  Procedure MST;{用 Prim 算法求最小生成树}
```

```
Var
  i,j:longint;
  temp:real;
  tb:rtype;
Begin
  Fillchar(f,Sizeof(f),0);
  For i:=1 to n Do
  Begin
    b[i].point:=i;
    If i=1 Then b[i].cost:=0 Else b[i].cost:=maxnum;
  End;
  For i:=1 to n-1 Do
  Begin
    For j:=i+1 to n Do
      If b[j].cost<b[i].cost Then
      Begin
        tb:=b[i]; b[i]:=b[j]; b[j]:=tb;
      End;
    For j:=i+1 to n Do
    Begin
      temp:=a[b[i].point,b[j].point];
      If temp<b[j].cost Then
      Begin
        b[j].cost:=temp;
        f[b[j].point]:=b[i].point;
      End;
    End;
  End;
End;
```

```
 Function Find(p,q,pos:longint):real;{在e[p]到e[q]中找到第pos大的元素}
Var
  r,i,j:longint;
  k:real;
Begin
  r:=Random(q-p+1)+p;
  k:=e[r];
  e[r]:=e[p];
  i:=p; j:=q;
  While i<j Do
  Begin
    While (i<j) and (k>=e[j]) Do Dec(j);
    e[i]:=e[j];
```

```
      While (i<j) and (e[i]>=k) Do Inc(i);
      e[j]:=e[i];
    End;
    e[i]:=k;
    If i=pos
      Then Find:=k
      Else If i<pos
            Then Find:=Find(i+1,q,pos)
            Else Find:=Find(p,i-1,pos);
  End;

  Procedure Answer;
  Var
    i:longint;
  Begin
    For i:=1 to n Do{把最小生成树中所有边的长度存放在e数组中}
      If i=1 Then e[i]:=0 Else e[i]:=a[i,f[i]];
    Assign(output,outputfile);
    Rewrite(output);
    Writeln(Find(1,n,k):0:2);{第k长的边}
    Close(output);
  End;

Begin
  Init;
  MST;
  Answer;
End.
```