



# 数据结构和算法

## (Python描述)

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

**学会程序和算法，走遍天下都不怕!**

讲义照片均为郭炜拍摄



北京大学  
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

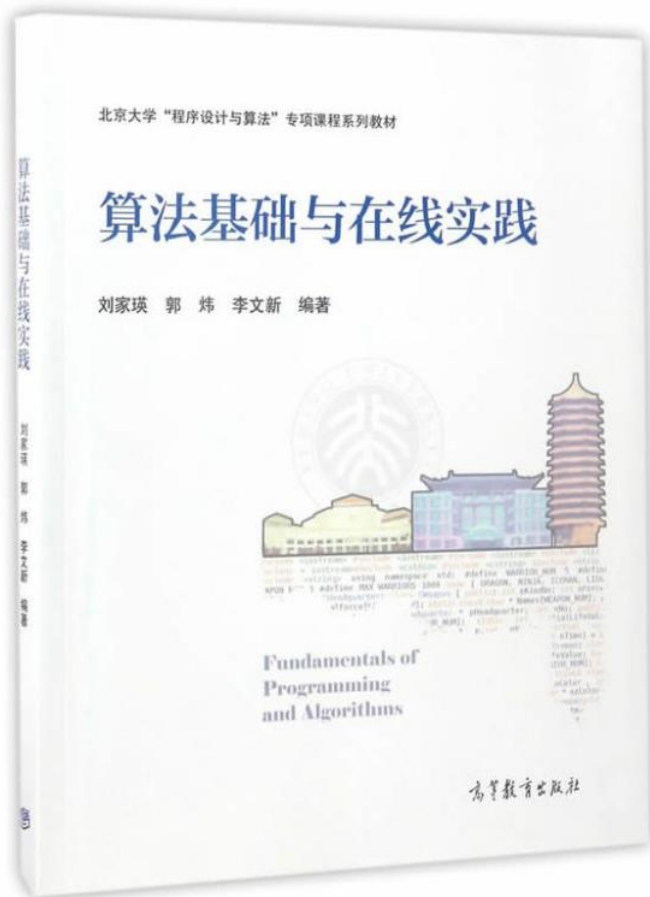
配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在  
<http://openjudge.cn>  
“百练”组进行搜索即可提交





# 最小生成树



北京大学  
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

## 图的生成树



河南云台山红石峡

# 图的生成树

- 在一个无向连通图 $G$ 中，如果取它的全部顶点和一部分边构成一个子图 $G'$ ，即：

$$V(G') = V(G); E(G') \subseteq E(G)$$

若边集 $E(G')$ 中的边既将图中的所有顶点连通又不形成回路，则称子图 $G'$ 是原图 $G$ 的一棵生成树。

- 一棵含有 $n$ 个点的生成树，必含有 $n-1$ 条边。

# 用深度优先遍求图的生成树

- 深度优先遍历一个图，记录每个顶点的父节点（即前驱节点，也即深度优先搜索树中的父节点），遍历结束后，将每个顶点及其父节点之间的边输出，即得到生成树。

# 用深度优先遍历求图的生成树

```
def dfsMst(G): #G连通无向图邻接表。求dfs生成树,顶点从0编号
    #G[s]就是s的邻点集合
    n = len(G)
    father = [-1 for i in range(n)]
    def dfs(v,f):
        father[v] = f
        for u in G[v]:
            if father[u] == -1: #没访问过
                dfs(u,v) #u的father是v
    dfs(0,0) #搜索树根节点父亲就是它自己
    result = [] #存放生成树的边
    for i in range(n):
        if father[i] != i:
            result.append((father[i],i))
    return result
```





北京大学  
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

# Prim算法 求最小生成树



河南郭亮村



# 最小生成树

- ✓ 对于一个无向连通带权图，每棵树的权（即树中所有边的权值总和）也可能不同
- ✓ 具有权最小的生成树称为最小生成树。

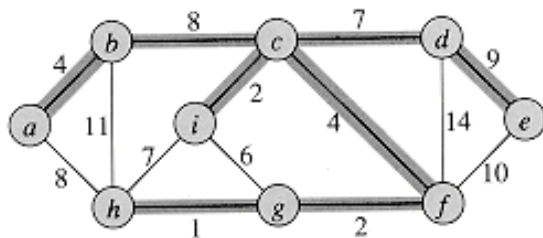
# 最小生成树

- 生成树

- 无向连通图的边的集合
- 无回路
- 连接所有的点

- 最小

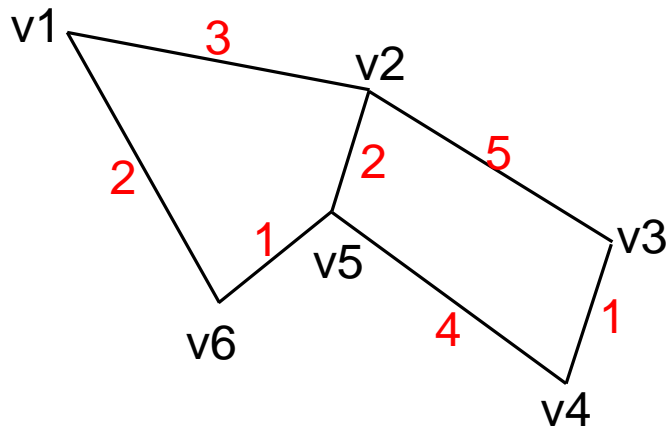
- 所有边的权值之和最小



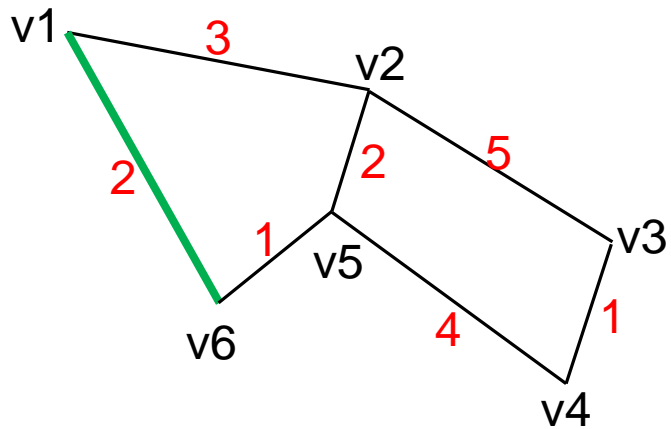
# Prim算法求最小生成树

- 假设 $G=(V,E)$ 是有 $n$ 个顶点的带权连通图,  $T=(U,TE)$ 是 $G$ 的最小生成树,  $U,TE$ 初值均为空集。
- 从 $V$ 中任取一个顶点将它并入 $U$ 中
- 每次从一个端点已在 $T$ 中, 另一个端点仍在 $T$ 外的所有边中, 找一条权值最小的, 并把该边及端点并入 $T$ 。做 $n-1$ 次,  $T$ 中就有 $n$ 个点,  $n-1$ 条边,  $T$ 就是最小生成树

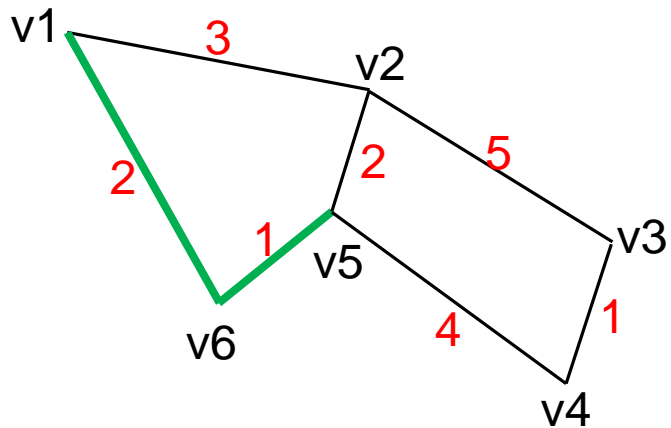
# Prim算法



# Prim算法

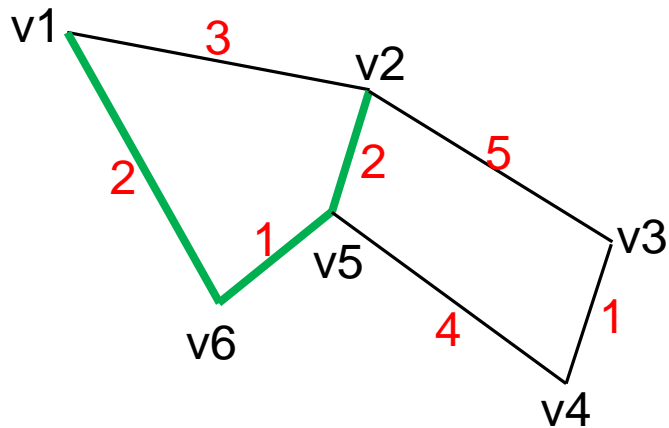


# Prim算法

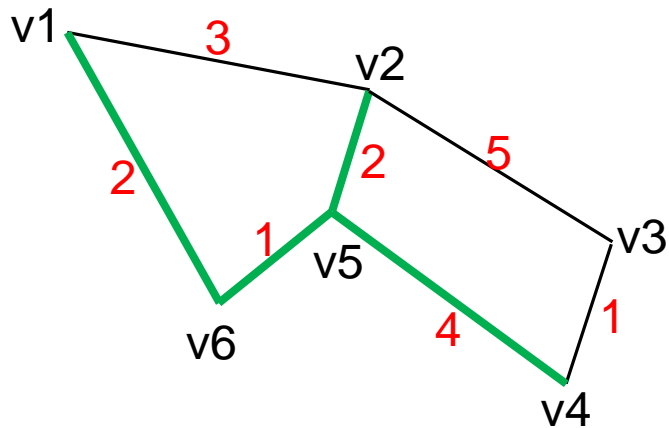




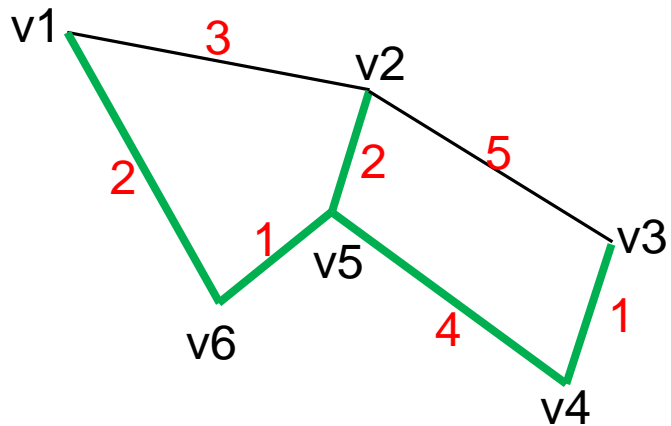
# Prim算法



# Prim算法



# Prim算法



# Prim算法实现

- 图节点数目为 $N$ ,正在构造的生成树为 $T$ ,
- 维护 $\text{Dist}$ 数组, $\text{Dist}[i]$ 表示 $V_i$ 到 $T$ 的“距离”,即 $V_i$ 和 $T$ 中所有的点的连边的最小权值
- 开始所有 $\text{Dist}[i] = \text{无穷大}$ ,  $T$  为空集

1) 若 $|T| = N$ , 最小生成树完成。否则取 $\text{Dist}[i]$ 最小的不在 $T$ 中的点 $V_i$ , 将其加入 $T$

2) **松弛操作**: 更新所有与 $V_i$ 有边相连且不在 $T$ 中的点 $V_j$ 的 $\text{Dist}$ 值:

$$\text{Dist}[j] = \min(\text{Dist}[j], W(V_i, V_j))$$

3) 转到1)

# Prim算法加快选边速度

## 每次如何从连接T中和T外顶点的所有边中，找到一条最短的

- ✓ 1) 如果用邻接矩阵存放图，而且选取最短边的时候遍历所有点进行选取，则总时间复杂度为 $O(V^2)$ (一共要选出 $V$ 个点，选出每个点时，选最小 $\text{dist}[i]$   $O(V)$ ，松弛操作 $O(V)$ )， $V$  为顶点个数
- ✓ 2) 用邻接表存放图,并使用堆来选取最小 $\text{dist}[i]$ ，则总时间复杂度为 $O(E \log V)$
- ✓ 不加堆优化的Prim 算法适用于密集图，加堆优化的适用于稀疏图

# POJ 1258 Agri-Net 最小生成树模版题

输入图的邻接矩阵，求最小生成树的总权值(多组数据)

输入样例：

4

0 4 9 21

4 0 8 17

9 8 0 16

21 17 16 0

输出样例：

28



# Prim + 堆 完成POJ1258 Agri-Net

```
import heapq
INF = 1 << 30

class Edge:
    def __init__(self, v=0, w=INF):
        self.v = v    #边端点, 另一端点已知
        self.w = w    #边权值或v到在建最小生成树的距离
    def __lt__(self, other):
        return self.w < other.w
```

```
def HeapPrim(G,n):  
    #G是邻接表,n是顶点数目,返回值是最小生成树权值和  
    xDist = Edge(0,0)  
    pq = []  
    heapq.heapify(pq) #存放顶点及其到在建生成树的距离  
    vDist = [INF for i in range(n)] #各顶点到已经建好的那部分树的距离  
    vUsed = [0 for i in range(n)] #标记顶点是否已经被加入最小生成树  
    doneNum = 0 #已经被加入最小生成树的顶点数目  
    totalW = 0 #最小生成树总权值  
    heapq.heappush(pq,Edge(0,0)) #开始只有顶点0, 它到最小生成树距离0
```

```
while doneNum < n and pq != []:
    while True: #每次从堆里面拿离在建生成树最近的不在生成树里面的点
        xDist = pq[0]
        heapq.heappop(pq)
        if not (vUsed[xDist.v] == 1 and pq != []):
            break
    if vUsed[xDist.v] == 0: #xDist.v要新加到生成树里面
        totalW += xDist.w
        vUsed[xDist.v] = 1
        doneNum += 1
        for i in range(len(G[xDist.v])): #更新新加入点的邻点
            k = G[xDist.v][i].v
            if vUsed[k] == 0:
                w = G[xDist.v][i].w
                if vDist[k] > w:
                    vDist[k] = w
                    heapq.heappush(pq, (Edge(k, w)))

if doneNum < n:
    return -1; #图不连通
return totalW
```

```
#main
```

```
while True:
    try:
        N = int(input())
        G = [[] for i in range(N)]
        for i in range(N):
            lst = list(map(int, input().split()))
            for j in range(N):
                G[i].append(Edge(j, lst[j]))
        print(HeapPrim(G, N))
    except:
        break
```

考察了所有的边，且考察一条边时可能执行  
`heapq.heappush(pq, (Edge(k, w)))`  
故复杂度 $O(E \log V)$



北京大学  
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

# Kruskal算法 求最小生成树



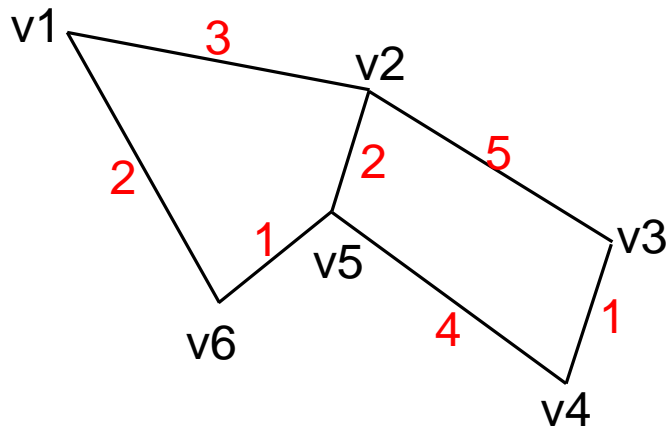
庐山如琴湖

# Kruskal算法

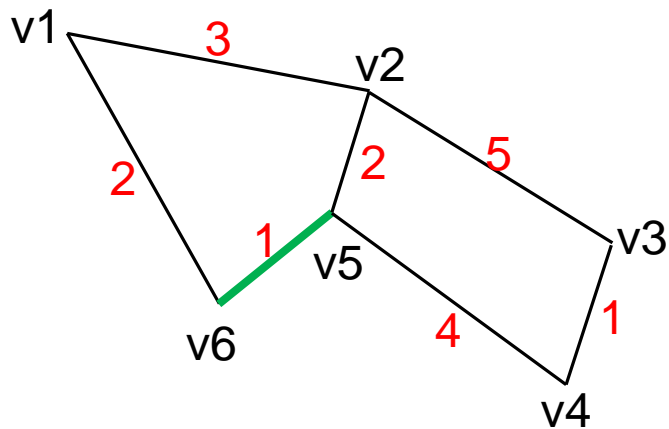
- 假设 $G=(V,E)$ 是一个具有 $n$ 个顶点的连通网， $T=(U,TE)$ 是 $G$ 的最小生成树， $U=V, TE$ 初值为空。
- 将图 $G$ 中的边按权值从小到大依次选取，若选取的边使生成树不形成回路，则把它并入 $TE$ 中，若形成回路则将其舍弃，直到 $TE$ 中包含 $N-1$ 条边为止，此时 $T$ 为最小生成树。



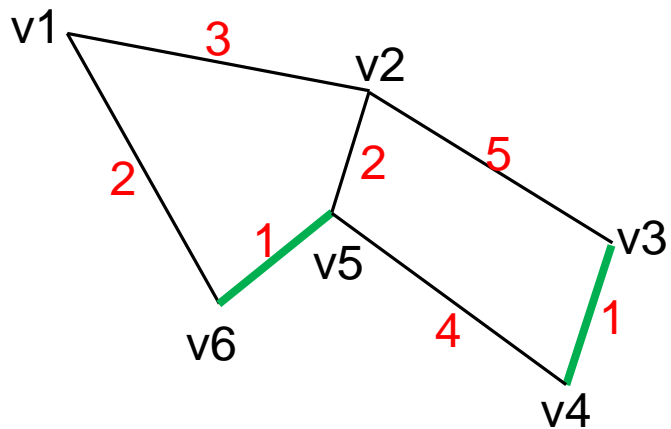
# Kruskal算法



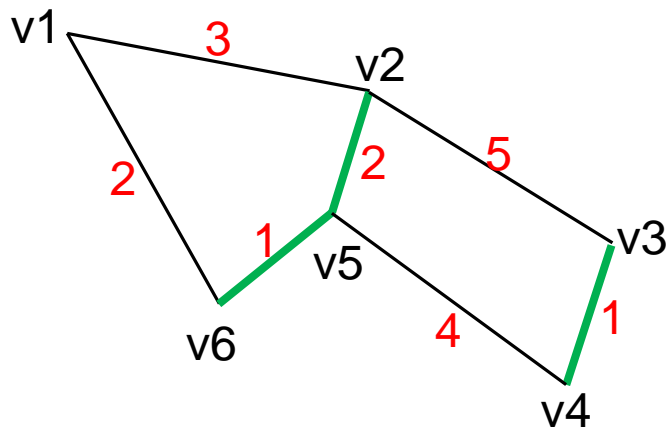
# Kruskal算法



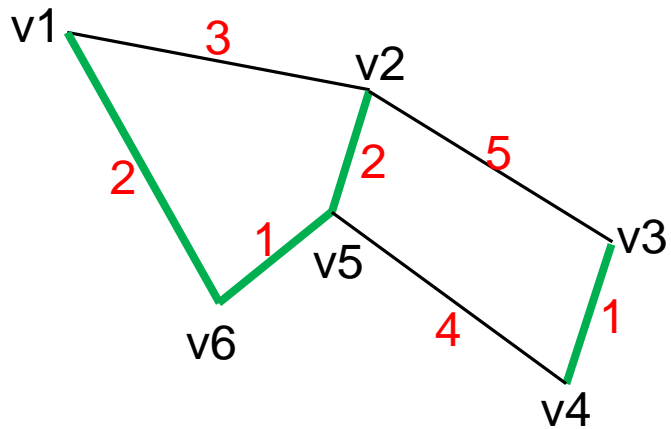
# Kruskal算法



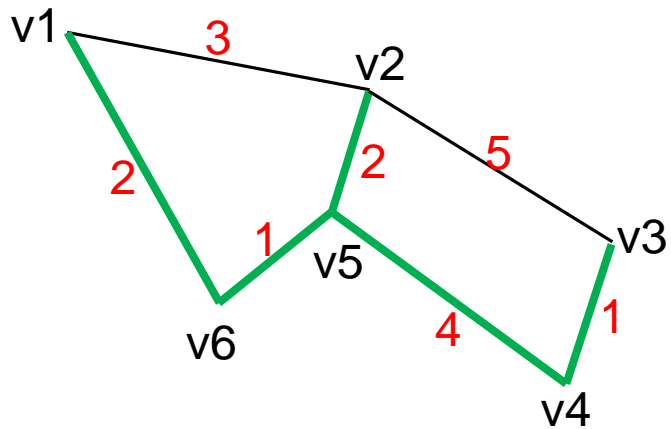
# Kruskal算法



# Kruskal算法



# Kruskal算法



# 关键问题

- ✓ **如何判断欲加入的一条边是否与生成树中边构成回路。**
- ✓ 将各顶点划分为所属集合的方法来解决，每个集合的表示一个无回路的子集。开始时边集为空， $N$ 个顶点分属 $N$ 个集合，每个集合只有一个顶点，表示顶点之间互不连通。
- ✓ 当从边集中按顺序选取一条边时，若它的两个端点分属于不同的集合，则表明此边连通了两个不同的部分，因每个部分连通无回路，故连通后仍不会产生回路，此边保留，同时把相应两个集合合并
- ✓ **要用并查集**提高效率

# Kruskal算法完成POJ1258 Agri-Net

```
INF = 1 << 30
```

```
class Edge:
```

```
    def __init__(self,s,e,w):
```

```
        self.s ,self.e,self.w = s,e,w    #起点, 终点, 权值
```

```
    def __lt__(self,other):
```

```
        return self.w < other.w
```

```
def GetRoot(a):
```

```
    if parent[a] == a:
```

```
        return a
```

```
    parent[a] = GetRoot(parent[a])
```

```
    return parent[a]
```

```
def Merge(a, b):
```

```
    p1 = GetRoot(a)
```

```
    p2 = GetRoot(b)
```

```
    if p1 == p2:
```

```
        return
```

```
    parent[p2] = p1
```



```
while True:    #main
```

```
    try:
```

```
        N = int(input())
```

```
        parent = [i for i in range(N)]
```

```
        edges = []
```

```
        for i in range(N):
```

```
            lst = list(map(int, input().split()))
```

```
            for j in range(N):
```

```
                edges.append(Edge(i, j, lst[j]))
```

```
edges.sort()    #排序复杂度 $O(E \log E)$ 
```

```
done = totalLen = 0
```

```
for edge in edges:
```

```
    if GetRoot(edge.s) != GetRoot(edge.e):
```

```
        Merge(edge.s, edge.e)
```

```
        done += 1
```

```
        totalLen += edge.w
```

```
    if done == N - 1:
```

```
        break
```

```
    print(totalLen)
```

```
except:    break
```

# 算法：Kruskal 和 Prim 的比较

- **Kruskal**: 将所有边从小到大加入，在此过程中判断是否构成回路
  - 使用数据结构：并查集
  - 时间复杂度： $O(E \log E)$
  - 适用于稀疏图
- **Prim**: 从任一节点出发，不断扩展
  - 使用数据结构：堆
  - 时间复杂度： $O(E \log V)$  或  $O(V \log V + E)$ (斐波那契堆)
  - 适用于密集图
  - 若不用堆则时间复杂度为 $O(V^2)$



北京大学  
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

## 例题



内蒙古阿斯哈图石林

## 例题： POJ 2349 Arctic Network

- 某地区共有 $n$ 座村庄，每座村庄的坐标用一对整数 $(x, y)$ 表示，现在要在村庄之间建立通讯网络。
- 通讯工具有两种，分别是需要铺设的普通线路和无线通讯的卫星设备。
- 只能给 $k$ 个村庄配备卫星设备，拥有卫星设备的村庄互相间直接通讯。
- 铺设了线路的村庄之间也可以通讯。但是由于技术原因，**两个村庄之间线路长度最多不能超过  $d$** ，否则就会由于信号衰减导致通讯不可靠。要想增大  $d$  值，则会导致要投入更多的设备（成本）

## 例题： POJ 2349 Arctic Network

- 已知所有村庄的坐标  $(x, y)$ ，卫星设备的数量  $k$ 。
- 问：如何分配卫星设备，才能使各个村庄之间能直接或间接的通讯，并且  $d$  的值最小？求出  $d$  的最小值。
- 数据规模：  $0 \leq k \leq n \leq 500$

## 思路

- 假设  $d$  已知，把所有铺设线路的村庄连接起来，构成一个图。需要卫星设备的台数就是图的连通支的个数。
- $d$  越小，连通支就可能越多。
- 那么，只需找到一个最小的  $d$ ，使得连通支的个数小于等于卫星设备的数目。

# 答案

把整个问题看做一个完全图，村庄就是点，图上两点之间的边的权值，就是两个村庄的直线距离。

只需在该图上求最小生成树， $d$  的最小值即为第  $K$  长边！

因为：最小生成树中的最长 $k-1$ 条长边都去掉后，正好将原树分成了 $k$ 个连通分量，在每个连通分量上摆一个卫星设备即可

那么剩下的各村庄之间的连线，最长不用超过 第  $k$  长边的长度。

## 为什么d不可能比第k长边更小？

假设最小生成树T上，第k长边连接的点是a,b，那么将边 $\langle a,b \rangle$ 去掉后，树就分成了两个部分T1 和T2

要使T1和T2能够通讯，必须在T1中找一点p和T2中的点q相连，若边 $\langle p,q \rangle$ 的长度小于 $\langle a,b \rangle$ ，则在T上用 $\langle p,q \rangle$ 替换 $\langle a,b \rangle$ 就能得到更小的生成树，矛盾。因此找不到长度小于 $\langle a,b \rangle$ 的 $\langle p,q \rangle$ 。

对任何比第k长边短的边e，同理也不可能找到替代e的边。

因此 d不可能更小了



最小生成树可能不止一棵，为什么第 $k$ 长边长度一定相同？因为有以下结论：

- 一个图的两棵最小生成树，边的权值序列排序后结果相同

证明：假设某个最小生成树 $T_1$ 的边权从小到大排序后的序列为：

$a_1, a_2, \dots, a_n$

某个最小生成树 $T_2$ 的边权从小到大排序后的序列为：

$b_1, b_2, \dots, b_n$

两者若不同，则必然存在一个最小的 $i$ ，使得  $a_i > b_i$

假设 $T_2$ 中有 $m$ 条边的权为 $b_i$ ，那么， $T_1$ 中最多只有 $m-1$ 条边的权和 $b_i$ 相同。

但是对于 $T_2$ 中任何一条不在 $T_1$ 中的权为 $b_i$ 的边，如果将其从 $T_2$ 去掉，则 $T_2$ 被分成 $A, B$ 两个部分。那么在 $T_1$ 中连接 $A, B$ 这两个部分的边，必然权值是等于 $b_i$ 的，否则经过替换，要么 $T_1$ 的权值可以变得更小，要么 $T_2$ 的权值可以变得更小，这和 $T_1, T_2$ 是最小生成树矛盾。对 $T_2$ 中每个权值为 $b_i$ 的边，都可以在 $T_1$ 中找到一个权值相同且不在 $T_2$ 的边与其对应，而这些边由于是连接不同部分的，所以不可能相同，因此，在 $T_1$ 中也应该有 $m$ 条权值为 $b_i$ 的边，这和 $T_1$ 中最多 $m-1$ 条权值为 $b_i$ 的边矛盾。因此，不存在 $i$ ，使得的 $a_i > b_i$ ，即两个边权序列应该相同。