

北京大学暑期课 《ACM/ICPC竞赛训练》

北京大学信息学院 郭炜

guo_wei@PKU.EDU.CN

<http://weibo.com/guoweiofpku>

课程网页: http://acm.pku.edu.cn/summerschool/pku_acm_train.htm

最小生成树 (MST) 问题

北京大学信息学院

郭炜/郑聃崴/陈国鹏

图的生成树

- 在一个连通图 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。

最小生成树

- ✓ 对于一个连通网（连通带权图，假定每条边上的权均为大于零的实数）来说，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。

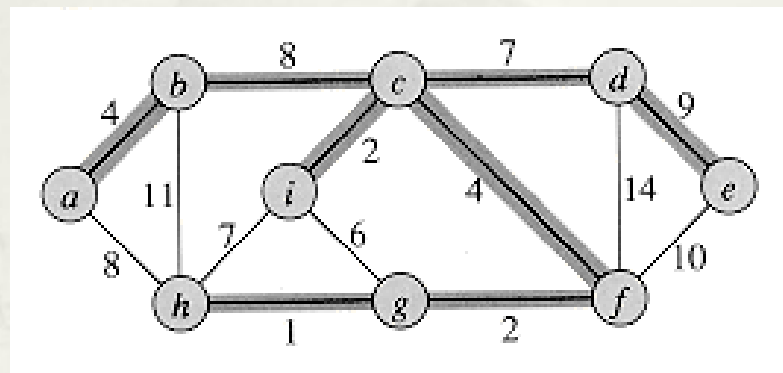
最小生成树

* 生成树

- * 无向连通图的边的集合
- * 无回路
- * 连接所有的点

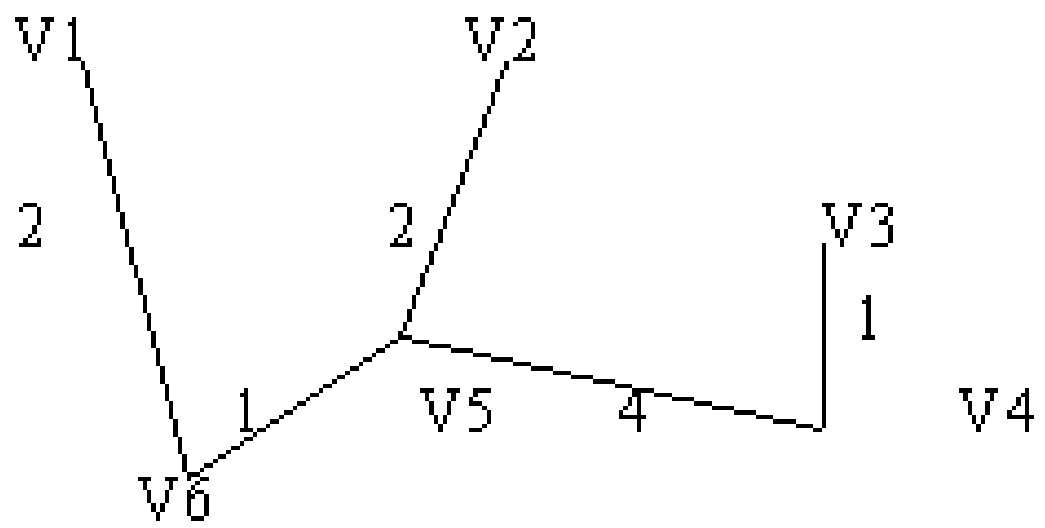
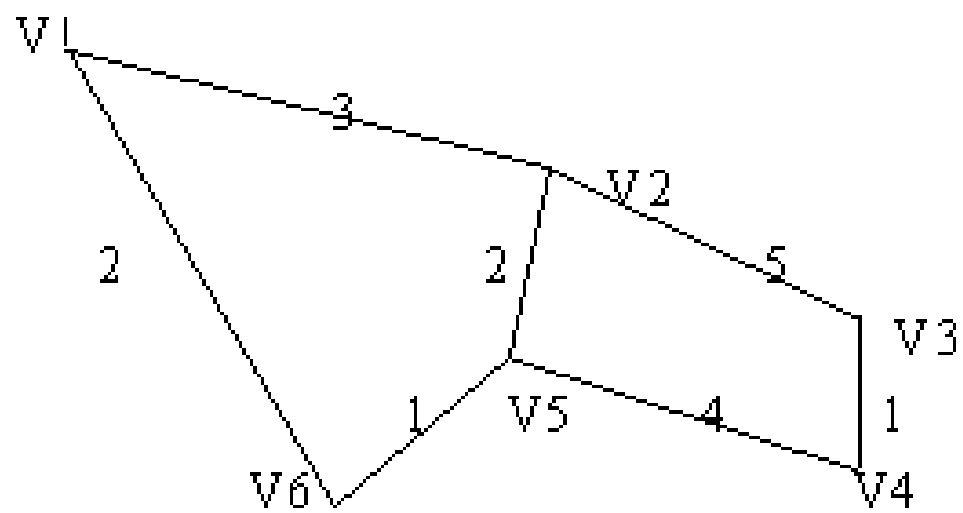
* 最小

- * 所有边的权值之和最小



Prim算法

- 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,TE 初值均为空集。
- 首先从 V 中任取一个顶点（假定取 v_1 ），将它并入 U 中，此时 $U=\{v_1\}$ ，然后只要 U 是 V 的真子集($U \subset V$)，就从那些一个端点已在 T 中，另一个端点仍在 T 外的所有边中，找一条最短边，设为 (v_i, v_j) ，其中 $v_i \in U, v_j \in V-U$ ，并把该边 (v_i, v_j) 和顶点 v_j 分别并入 T 的边集 TE 和顶点集 U ，如此进行下去，每次往生成树里并入一个顶点和一条边，直到 $n-1$ 次后得到最小生成树。



Prim算法实现

- 图节点数目为 N ,正在构造的生成树为 T ,
 - 维护Dist数组, $\text{Dist}[i]$ 表示 V_i 到 T 的“距离”
 - 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) 更新所有与 V_i 有边相连且不在 T 中的点 V_j 的Dist值:
$$\text{Dist}[j] = \min(\text{Dist}[j], W(V_i, V_j))$$
 - 3) 转到1)

关键问题

- ✓ 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 $O(V^2)$, V 为顶点个数
- ✓ 2) 用邻接表存放图, 并使用堆来选取最短边，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 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

priority_queue实现 Prim + 堆 完成POJ1258

//by Guo Wei

```
#include <iostream>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
#include <queue>
```

```
using namespace std;
```

```
const int INFINITE = 1 << 30;
```

```
struct Edge
```

```
{
```

```
    int v; //边端点，另一端点已知
```

```
    int w; //边权值，也用来表示v到在建最小生成树的距离
```

```
    Edge(int v_ = 0, int w_ = INFINITE):v(v_),w(w_) {}
```

```
    bool operator <(const Edge & e) const
```

```
    {
```

```
        return w > e.w; //在队列里，边权值越小越优先
```

```
    }
```

```
};
```

```
vector< vector <Edge> > G(110); //图的邻接表
```

```
int HeapPrim(const vector<vector<Edge> > & G, int n)
//G是邻接表,n是顶点数目, 返回值是最小生成树权值和
{
    int i,j,k;
    Edge xDist(0,0);
    priority_queue<Edge> pq; //存放顶点及其到在建生成树的距离
    vector<int> vDist(n); //各顶点到已经建好的那部分树的距离
    vector<int> vUsed(n); //标记顶点是否已经被加入最小生成树
    int nDoneNum = 0; //已经被加入最小生成树的顶点数目
    for( i = 0; i < n; i ++ ) {
        vUsed[i] = 0;
        vDist[i] = INFINITE;
    }
    nDoneNum = 0;
    int nTotalW = 0; //最小生成树总权值
    pq.push(Edge(0,0)); //开始只有顶点0, 它到最小生成树距离0
```

```

while( nDoneNum < n && !pq.empty() ) {
    do { //每次从队列里面拿离在建生成树最近的点
        xDist = pq.top();          pq.pop();
    } while( vUsed[xDist.v] == 1 && ! pq.empty());
    if( vUsed[xDist.v] == 0 ) {
        nTotalW += xDist.w;  vUsed[xDist.v] = 1; nDoneNum ++;
        for( i = 0; i < G[xDist.v].size(); i ++ ) { //更新新加入点的邻点
            int k = G[xDist.v][i].v;
            if( vUsed[k] == 0 ) {
                int w = G[xDist.v][i].w ;
                if( vDist[k] > w ) {
                    vDist[k] = w;
                    pq.push(Edge(k,w));
                }
            }
        }
    }
}
if( nDoneNum < n )
    return -1; //图不连通
return nTotalW;
}

```

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

```
int main()
{
    int N;
    while(cin >> N) {
        for( int i = 0; i < N; ++i)
            G[i].clear();
        for( int i = 0; i < N; ++i)
            for( int j = 0; j < N; ++j) {
                int w;
                cin >> w;
                G[i].push_back(Edge(j,w));
            }
        cout << HeapPrim(G,N) << endl;
    }
}
```

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。

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并

Kruskal算法完成POJ1258

//by Guo Wei

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Edge
{
    int s,e,w; //起点， 终点， 权值
    Edge(int ss,int ee,int ww):s(ss),e(ee),w(ww) {}
    Edge() {}
    bool operator < (const Edge & e1) const {
        return w < e1.w;
    }
};
vector <Edge> edges;
vector <int> parent;
```

```
int GetRoot(int a)
{
    if( parent[a] == a)
        return a;
    parent[a] = GetRoot(parent[a]);
    return parent[a];
}

void Merge(int a,int b)
{
    int p1 = GetRoot(a);
    int p2 = GetRoot(b);
    if( p1 == p2)
        return;
    parent[p2] = p1;
}
```

```
int main() {
    int N;
    while(cin >> N) {
        parent.clear();    edges.clear();
        for( int i = 0; i < N; ++i) parent.push_back(i);
        for( int i = 0; i < N; ++i)
            for( int j = 0; j < N; ++j) {
                int w;
                cin >> w;
                edges.push_back(Edge(i,j,w));
            }
        sort(edges.begin(),edges.end()); //排序复杂度O(ElogE)
        int done = 0;    int totalLen = 0;
        for( int i = 0; i < edges.size(); ++i) {
            if( GetRoot(edges[i].s) != GetRoot(edges[i].e)) {
                Merge(edges[i].s,edges[i].e);
                ++done;
                totalLen += edges[i].w;
            }
            if( done == N - 1) break;
        }
        cout << totalLen << endl;
    }
}
```

算法：Kruskal 和 Prim

- **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 $O(V^2)$

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： $0 \leq k \leq n \leq 500$

(From Waterloo University 2002)

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 个连通分支，在每个连通分支上摆一个卫星设备即可

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**T1**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**T2**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**T2**中有 m 条边的权为 b_i ，那么，**T1**中最多只有 $m-1$ 条边的权和 b_i 相同。

但是对于**T2**中任何一条不在**T1**中的权为 b_i 的边，如果将其从**T2**去掉，则**T2**被分成A,B两个部分。那么在**T1**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**T1**的权值可以变得更小，要么**T2**的权值可以变得更小，这和**T1**,**T2**是最小生成树矛盾。对**T2**中每个权值为 b_i 的边，都可以在**T1**中找到一个权值相同且不在**T2**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**T1**中也应该有 m 条权值为 b_i 的边，这和**T1**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。

2011 ACM/ICPC亚洲区预选赛北京赛站

Problem A. Qin Shi Huang's National Road System

一个无向完全图，边有正权值，点也有正权值。可以选择一条边，将其边权值变为0。要求选定这条边(假定为 e_0)并将其权值变为0后，满足以下条件： A/B 最大。其中 A 是 e_0 连接的两个点的点权值和， B 是修改后的图的最小生成树的边权值和。

解题思路：先求一棵最小生成树，求的过程中，每加入一个点，就记录已经在树上的所有点到该点的路径（树上的路径）上的最长边的权值。然后枚举权值要变成0的边 uv ，如果 uv 不是树边，则用它替换 uv 路径上的最大权值边， $O(1)$ 时间即得新最小生成树的边权值和； uv 是树边，新最小生成树的边权值和即为原最小生成树的边权值和减去边 uv 的权值。

需要知道结点 u 到 v 的最长边

$dp[u][s] = \max(dp[u][v], dp[v][s])$

- 红色部分的做法：
- prim算法中，已经加入生成树的点集合为W
- 往W新增点s时，设 u 属于W,且 s 是被连接到W中的v点的，
- 则
 - $\text{Max_val}[v][s] = \text{边}(v,s)\text{的权}$
 - $\text{Max_val}[u][s] = \text{Max}(\text{Max_val}[v][s], \text{Max_val}[u][v])$
- 用时 $O(V^2)$ 。