POJ 3164 最小树形图 朱刘算法

个人觉得这个博客把这个算法说的比较详细了，直接搬过来吧，我再阐述一遍的话没有人家说的好，还容易说错。

========== 分 割 线 之 下 摘 自 Sasuke_SCUT 的 blog==========

最 小树形图，就是给有向带权图中指定一个特殊的点 root，求一棵以 root 为根的有向生成树 T，并且 T 中所有边的总权值最小。最小树形图的第一个算法是 1965年朱永津和刘振宏提出的复杂度为 O(VE)的算法。

判断是否存在树形图的方法很简单，只需要以 v 为根作一次图的遍历就可以了，所以下面的算法中不再考虑树形图不存在的情况。

在所有操作开始之前，我们需要把图中所有的自环全都清除。很明显，自环是不可能在任何一个树形图上的。只有进 行了这步操作，总算法复杂度才真正能保证是 O(VE)。

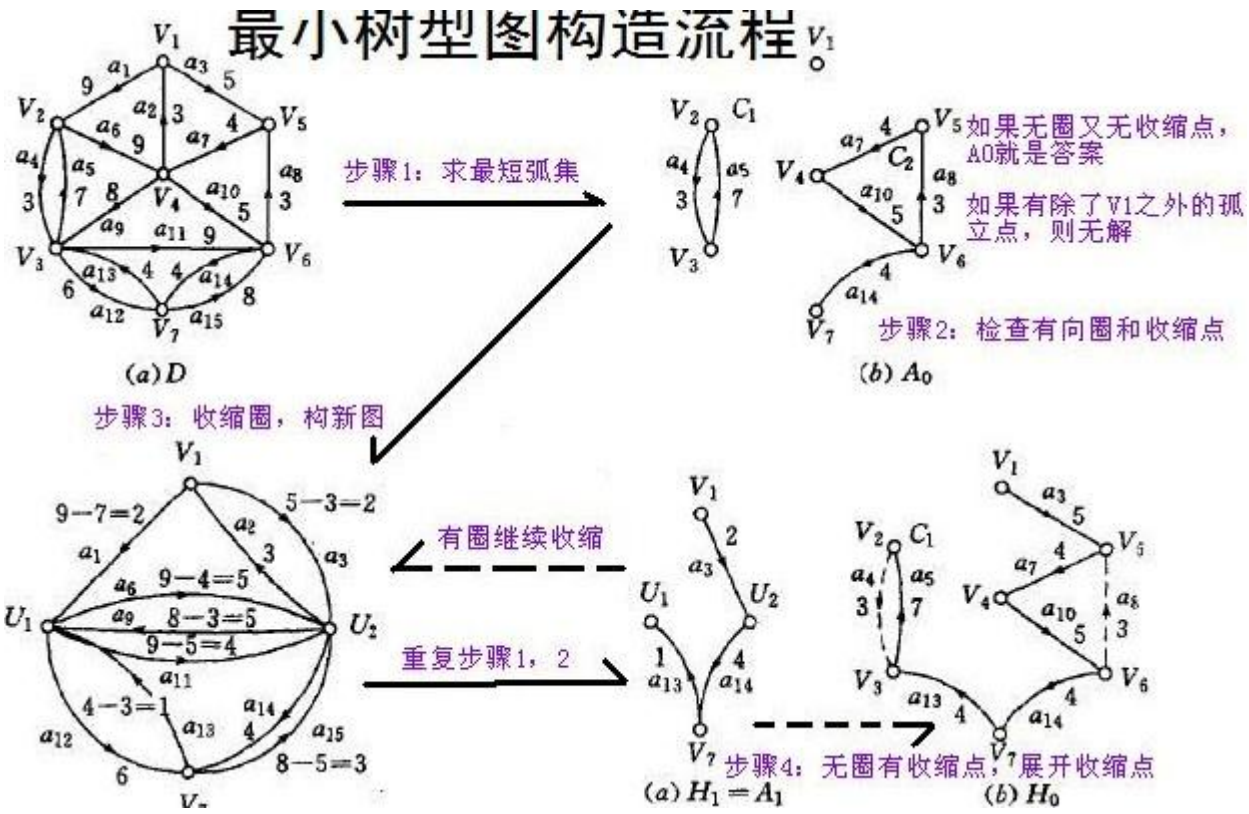
首先为除根之外的每个点选定一条入边，这条入边一定要是所有入边中最小的。现在所有的最小 入边都选择出来了，如果这个入边集不存在有向环的话，我们可以证明这个集合就是该图的最小树形图。这个证明并不是很难。如果存在有向环的话，我们就要将这 个有向环所称一个人工顶点，同时改变图中边的权。假设某点 u 在该环上，并设这个环中指向 u 的边权是 in[u]，那么对于每条从 u 出发的边(u, i, w)，在新图中连接(new, i, w)的边，其中 new 为新加的人工顶点; 对于每条进入 u 的边(i, u, w)，在新图中建立边(i, new, w-in[u])的边。为什么入边的权要减去 in[u]，这个后面会解释，在这里先给出算法的步骤。然后可以证明，新图中最小树形图的权加上旧图中被收缩 的那个环的权和，就是原图中最小树形图的权。

上面结论也不做证明了。现在依据上面的结论，说明一下为什么出边的权不变，入边的权要减去 in [u]。对于新图中的最小树形图 T，设指向人工节点的边为 e。将人工节点展开以后，e 指向了一个环。假设原先 e 是指向 u 的，这个时候我们将环上指向 u 的边 in[u]删除，这样就得到了原图中的一个树形图。我们会发现，如果新图中 e 的权 w'(e)是原图中 e 的权 w(e)减去 in[u]权的话，那么在我们删除 掉 in[u]，并且将 e 恢复为原图状态的时候，这个树形图的权仍然是新图树形图的权加环的权，而这个权值正是最小树形图的权值。所以在展开节点之后，我们 得到的仍然是最小树形图。逐步展开所有的人工节点，就会得到初始图的最小树形图了。

如果实现得很聪明的话，可以达到找最小入边 O(E)，找环 O(V)，收缩 O(E)，其中在找环 O(V)这里需要一点技巧。这样每次收缩的复杂度是 O(E)，然后最多会收缩几次呢？由于我们一开始已经拿掉了所有的 自环，我门可以知道每个环至少包含2个点，收缩成1个点之后，总点数减少了至少1。当整个图收缩到只有1个点的时候，最小树形图就不不用求了。所以我们最 多只会进行 V-1次的收缩，所以总得复杂度自然是 O(VE)了。由此可见，如果一开始不除去自环的话，理论复杂度会和自环的数目有关。

========== 分 割 线 之 上 摘 自 Sasuke_SCUT 的 blog==========

下 面是朱刘算法的构造图



下面是 POJ 3164的代码

```
#include<iostream>
#include<cmath>
#define INF 1000000000
using namespace std;
double map[110][110];
bool visit[110],circle[110];
int pre[110];
int n,m;
struct PT
{
    double x,y;
}p[110];
double dist(int i,int j)
{
    return
sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)+(p[i].y-p[j].y)*(p[i].y-p[j].y));
}
void dfs(int t)
{
    if(visit[t])
```

```
        return ;
    visit[t]=1;
    for(int i=1;i<=n;i++)
        if(map[t][i]<INF)
            dfs(i);
}
bool connect()//深搜，判断是否存在最小树形图
{
    dfs(1);
    for(int i=1;i<=n;i++)
        if(!visit[i])
            return 0;
    return 1;
}
double solve()
{
    double ans=0;
    int i,j,k;
    memset(circle,0,sizeof(circle));//如果某点被删掉，那么 circle[i]=1
    while(1)
    {
        for(i=2;i<=n;i++)//求出每个点的最小入边
        {
            if(circle[i])
                continue;
            map[i][i]=INF;
            pre[i]=i;
            for(j=1;j<=n;j++)
            {
                if(circle[j])
                    continue;
                if(map[j][i]<map[pre[i]][i])
                    pre[i]=j;
            }
        }
        for(i=2;i<=n;i++)//遍历找环
        {
            if(circle[i])
                continue;
            j=i;
            memset(visit,0,sizeof(visit));
            while(!visit[j]&&j!=1)
            {
                visit[j]=1;
```

```
            j=pre[j];
        }
        if(j==1)//j==1说明 i 不在环上
            continue;
        i=j;//找到了环
        ans+=map[pre[i]][i];
        for(j=pre[i];j!=i;j=pre[j])
        {
            ans+=map[pre[j]][j];
            circle[j]=1;//用环上一点 i 代表此环，其他点删去，即 circle[j]=1
        }
        for(j=1;j<=n;j++)
        {
            if(circle[j])
                continue;
            if(map[j][i]<INF)
                map[j][i]-=map[pre[i]][i];//更新 j 的入边
        }
        for(j=pre[i];j!=i;j=pre[j])//环上所有点的最优边为人工顶点的边
        {
            for(k=1;k<=n;k++)
            {
                if(circle[k])
                    continue;
                if(map[j][k]<INF)
                    map[i][k]=min(map[i][k],map[j][k]);
                if(map[k][j]<INF)
                    map[k][i]=min(map[k][i],map[k][j]-map[pre[j]][j]);
            }
        }
        break;
    }
    if(i>n)
    {
        for(j=2;j<=n;j++)
        {
            if(circle[j])
                continue;
            ans+=map[pre[j]][j];
        }
        break;
    }
}
return ans;
```

```
}
int main()
{
    int i,j;
    int a,b;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        for(i=1;i<=n;i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                map[i][j]=INF;
        for(i=1;i<=m;i++)
        {
            scanf("%d%d",&a,&b);
            map[a][b]=dist(a,b);
        }
        memset(visit,0,sizeof(visit));
        if(!connect())
            printf("poor snoopy\n");
        else printf("%.2lf\n",solve());
    }
        return 0;
}
```