西安邮电大学第十五届

数学建模竞赛参赛作品

参赛队编号： 004

赛题类型代码： A

自来水管道铺设规划

摘 要：

自来水是人们日常生活中不可缺少的生活要素，在自来水工程实施的过程中，为节水以及方便生活，需要在各级供水站间按照题目要求铺设两种型号的自来水管道，同时需要保证每个供水站都有与其他供水站间铺设自来水管道，并且不能造成自来水管道资源的浪费。

问题 1 的回答：自来水管道将各个供水站用最短路径连接是一个典型的最小生成树问题，先在 Dec-C++上编写程序，得出相邻供水站间的距离数据，再采用 Kruskal 算法建立模型，输入所得出的数据，最终通过 MATLAB-2016b 编程、绘图，用图形展现管道铺设的情况。

问题 2 的回答：为减少 II 型管道铺设的总里程，确定需要升级的两个二级供水站，不能与其他二级供水站之间铺设自来水管道，同时也不能影响其他供水站间管道的铺设。在分叉树末端的二级供水站与上一个二级供水站间只铺设一条 II 型管道，若将该二级供水站升级为一级供水站，再与邻近的一级供水站间铺设管道，这样就减少 II 型管道铺设的总里程。先写程序对末端铺设 II 型管道的里程进行筛选，后取最大的两个数值所对应的二级供水站升级为一级供水站即可。

问题 3 的回答：至少升级几个二级供水站后，在它自身承受 40 公里以内压力的情况下，能使自来水管道设的总里程最少。采用树和图的思想来思考问题，建立多叉树模型，运用并查集算法编写程序，寻找子节点并计算它们之间的距离；汇总已给的 12 个一级供水站铺设的总里程，对于里程压力超过 40 公里的供水站，在它附近寻找二级供水站进行升级，在给该供水站升级后分压里程的同时也减少了所铺设管道的总里程。

关键字：最小生成树；Kruskal 算法；多叉树模型；并查集算法

一、问题重述

自来水是人们日常生活中不可缺少的生活要素，在自来水工程实施的过程中，为节水以及方便生活，需要在 181 个供水站之间铺设管道（分别为 1 个中心供水站，12 个一级供水站和 168 个二级供水站），对于铺设管道的要求是：

（1）供水站之间所铺设的管道必须从上一级供水站或同一级供水站的位置坐标出发，到达其连接的另一个供水站，不能从任意管道中间产生分支进行连接。

（2)I 型管道只能在中心供水站与一级供水站以及一级供水站之间进行铺设；

（3)II 型管道只能在一级供水站与二级供水站以及二级供水站之间进行铺设；

（4）铺设管道的长度是所连接两个供水站之间的直线距离。

问题 1：从中心供水站 A 开始铺设使得管道铺设的总里程最小，以图形的形式给出自来水管道的铺设方案，并解得各型管道总里程数。

问题 2：对于所铺设 II 型管道的图形进行分析，在 168 个二级供水站中选择哪个两个升级为一级供水站，从而减少所铺设 II 型管道的总里程，同时跟问题 1 的结果进行比较，看 II 型管道总里程减少的数值。

问题 3：增加限制条件，要求从一级供水站（包括二级供水站升级成一级供水站的）出发铺设的管道受距离限制，最多只能供水 40 公里，但从中心供水站 A 出发铺设的管道供水不受此距离限制。当能对所有的供水站进行供水时，升级的二级供水站的最少数量，并解答在该条件下铺设自来水管道的最少总里程。

二、问题分析

问题 1 的分析：

自来水管道将各个供水站用最短路径连接是一个典型的最小生成树问题。在 181 个供水站之间分级铺设管道，先对 1 个中心供水站和 12 个一级供水站采用 Kruskal 算法建立模型，得到 I 型管道铺设最少的方案，在此基础上对 12 个一级供水站和 168 个二级供水站采用 Kruskal 算法建立模型，得到 II 型管道铺设最少的方案，最终通过 MATLAB-2016b 编程、绘图，用图形展现管道铺设的情况，解决该问题。

问题 2 的分析：

若减少从一级供水站出发铺设的 II 型管道总里程，则该二级供水站在升级后，

不能与二级供水站之间铺设管道，同时也不能影响其他供水站间管道的铺设。若从中心供水站 A 出发，在中部任选一个二级供水站进行升级后，由于它还需与下一个二级供水站间铺设 II 型管道，这样并没有完成题目要求。对问题一得到的分叉、无向连通的最小生成树进行分析，在树枝末端的二级供水站与上一个二级供水站间只铺设一条 II 型管道，若将该二级供水站升级为一级供水站，再与临近的一级供水站间铺设管道，这样就减少 II 型管道铺设的总里程。先写程序对末端铺设 II 型管道的里程进行筛选，后取最大的两个数值所对应的二级供水站升级为一级供水站即可，此时 II 型管道铺设总里程的减少量就是上述筛选的两个最大里程数值的和。

问题 3 的分析：

当增加约束条件 40 公里后，至少升级几个二级供水站后，在它自身承受 40 公里以内压力的情况下，能使自来水管道设的总里程最少。由于数据较多，情况较为复杂，要求在问题 1 的基础上，即不改变问题 1 中 I 型管道、II 型管道铺设的整体布局，采用树和图的思想来思考问题，建立多叉树模型，运用并查集算法编写程序，寻找子节点并计算它们之间的距离；同时汇总已给出的 12 个一级供水站铺设的总里程，对于总里程压力超过 40 公里的一级供水站，需要在它附近寻找二级供水站进行升级分担该一级供水站的里程压力。我们的目标是在尽可能少升级二级供水站的情况下，使每个一级供水站承受里程的压力小于或等于 40 公里，同时使减少的自来水管道总里程数最多。

三、模型假设与符号说明

3.1 模型假设

1. 供水站都能通过自来水管道获得自来水供应；
2. 文中给出所有点的坐标值都准确无误；
3. 所需要铺设管道的长度是两供水站之间的直线距离；
4. 假设任何两个供水站之间都可以直线连接（除了中心供水站）。

3.2 符号说明

符号	符号说明

sum	铺设管道的总里程
$sumA$	铺设 I 型管道的总里程
$sumB$	铺设 II 型管道的总里程
d	任意两个供水站间的距离
(x_1,y_1)、(x_2,y_2)	任意两个供水站的坐标
n	一级供水站的总数量
m	二级供水站的总数量
$\Delta sumB$	II 型管道减少的里程
Δsum	自来水管道减少的总里程

四、模型建立与求解

4.1 模型准备

4.1.1.两点间的距离

$$d=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

运行程序 E_1.cpp.

4.1.2.最小生成树[1]：

在一给定的无向图 G = (V, E) 中，(u, v) 代表连接顶点 u 与顶点 v 的边（即），而 w(u, v) 代表此边的权重，若存在 T 为 E 的子集（即）且为无循环图，使得 w(T) 最小，则此 T 为 G 的最小生成树。自来水管道将各个供水站用最短路径连接是一个典型的最小生成树问题，在 181 个供水站间得到一个分叉、无向连通图，其间不形成环路，即为加权的最小生成树。采用 Kruskal 算法[2]，总共选择 n- 1 条边，(共 n 个点)所使用的贪心准则是:从剩下的边中选择一条不会产生环路的具有最小耗费的边加入已选择的边的集合中，若所选取的边若产生环路则不可能形成一棵生成树。

运行程序 E_2.cpp.

4.2 问题 1 模型建立与求解

4.2.1.I 型管道、II 型管道的铺设路线：

通过 MATLAB-2016b[3]运行程序 E_3.m 得到：

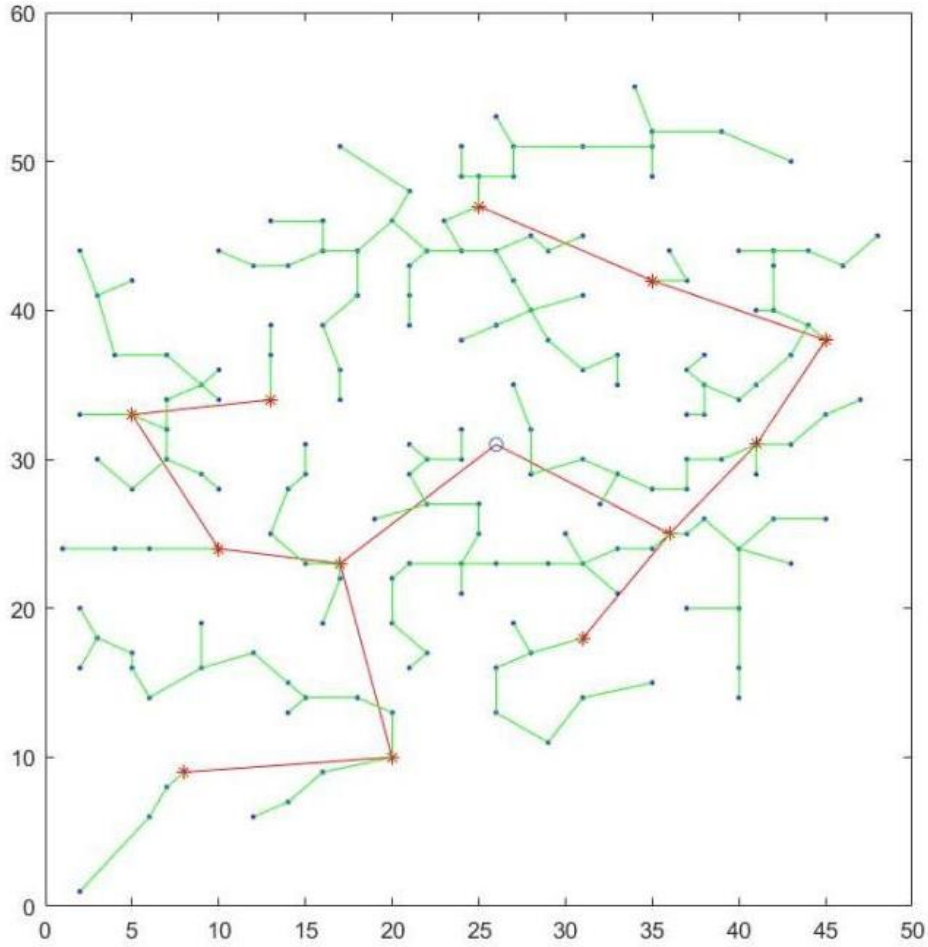

图 1 I 型管道（红）、II 型管道（绿）的铺设图形

4.2.2.自来水管道 I 型管道、II 型管道及总体铺设的里程：

$$sumA=\sum_{i=1}^{n} d_i$$

$$sumB=\sum_{j=1}^{m} d_j$$

$$sum = sumA + sumB$$

运行程序 E_3.m 得到：

$$sumA = 120.9412$$
$$sumB = 403.6408$$
$$sum = 524.582$$

4.3 问题 2 模型建立与求解

4.3.1.给图 1 中的点注明类型：

运行程序 E_4.m 得到：

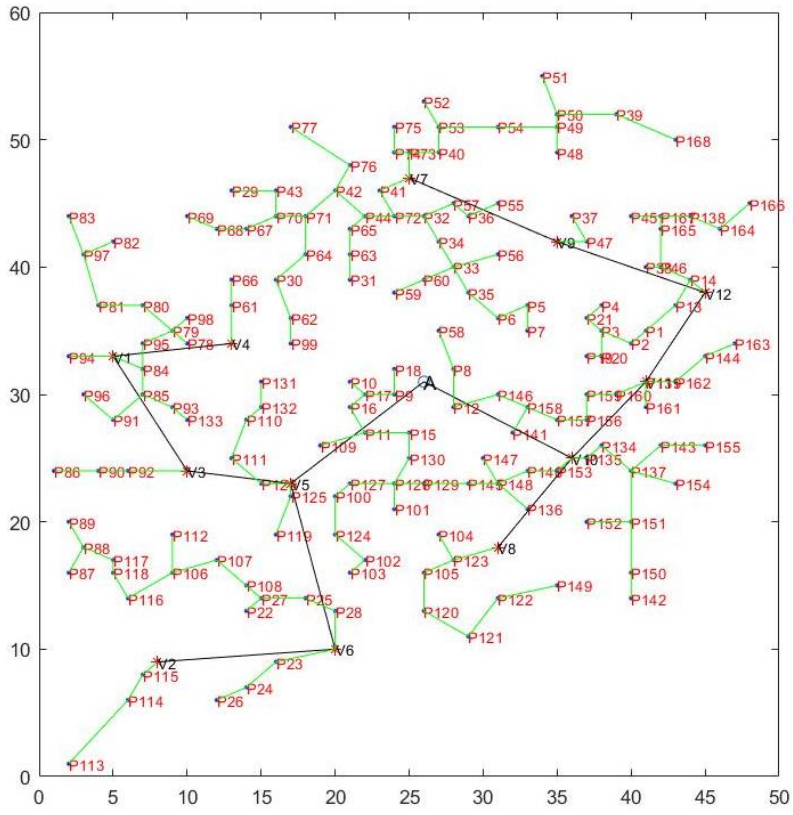

图 2 标明类型图

4.3.2.对末端铺设 II 型管道的里程进行筛选：

运行程序 E_5.c 得到：

表 1 筛选结果

类型	里程(公里)	类型	里程(公里)
P113	6.40312	P59	2.23607
P77	5.00000	P147	2.23607
P168	4.47214	P104	2.23607
P149	4.12311	P69	2.23607
P109	3.16228	P163	2.23607
P56	3.16228	P142	2.00000
P58	3.16228	P48	2.00000
P119	3.16228	P99	2.00000
P83	3.16228	P101	2.00000
P51	3.16228	P18	2.00000
P154	3.16228	P66	2.00000
P112	3.00000	P31	2.00000
P86	3.00000	P131	2.00000
P29	3.00000	P161	2.00000
P94	3.00000	P7	2.00000
P152	3.00000	P45	2.00000
P155	3.00000	P75	2.00000
P136	2.82843	P133	1.41421
P96	2.82843	P103	1.41421
P166	2.82843	P78	1.41421
P82	2.23607	P22	1.41421
P52	2.23607	P10	1.41421
P87	2.23607	P98	1.41421
P89	2.23607	P4	1.41421
P55	2.23607	P19	1.00000
P37	2.23607	P59	2.23607
P26	2.23607	P147	2.23607
P141	2.23607		

4.3.3.确定要升级的二级供水站：

表 1 中的里程数是按照降序排列的，取前两个里程数所对应的类型点：

表 2 需升级二级供水站结果

类型	里程(公里)
P113	6.40312

P77	5.00000

即升级 P113、P77 对应的两个二级供水站，与就近的一级供水站间铺设 I 型管道，如 P113 与 V2、P77 与 V7 间分别铺设 I 型管道，这样 II 型管道总里程达到最少，较问题 1 比，II 型管道的总里程减少了 $\Delta sumB$ 公里：

$$\Delta sumB = 11.40312$$

4.4 问题 3 模型建立与求解

4.4.1 对于该问题的求解我们需要知道已给出的 12 个一级供水站所铺设 I 型管道的总里程，各二级供水站与一级供水站的距离以及从中心供水站 A 出发后各个供水站的子节点及其距离。采用树和图的思想，建立多叉树模型[4]，运用并查集算法[5]编写 E_6.c 的程序，并在 Dec-C++上运行，得到附件一、二、三的结果。

4.4.2 在附件二中，对于大于 40 公里的数据进行筛选后，标红的单元格数据对应的一级供水站，需要在其周围寻找二级供水站进行升级，替它分压里程，分析结果如下表(“-”为减少，“+”为增加；单位：公里)：

表 2 改造结果

需改造一级供水站	改造方案	一型管道	二型管道	总里程
V1	升级 P78	0.03676	-2.23607	-2.19931
V6	升级 P22	-3.00000	1.87771	-1.12229
V10	升级 P9	0.09396	-2.00000	-1.90604
	升级 P134	0.25677	-1.41421	-1.15744
V11、V12	升级 P19	4.47214	-2.06450	2.40764
V7	升级 P56	1.42809	-2.00000	-0.57191
	升级 P99	4.00000	-2.82843	1.17157
总计		5.85963	-10.66550	-2.66235

综合分析，确定升级 7 个二级供水站，在保证对所有供水站供水的情况下也减少了 I 型管道、II 型管道铺设的总里程，减少的总里程为 Δsum 公里：

$$\Delta sum = 2.66235$$

五、模型评价

5.1 模型优缺点

5.1.1 模型优点

(1)本题的数据比较多，最小生成树较为稠密，而 Kruskal 算法适用于稠密树，绘制出的图形清楚地表达了结果，故采用 Kruskal 算法建立的模型。

(2)多叉树模型过程不是太过于复杂，能够处理目前所有的数据。

(3)并查集是一种树型的数据结构，用于处理一些不相交集合的合并及查询问题。

(4)运行该模型进行最小生成树的生成可以一步到位，不需要人工修改。

5.1.2 模型缺点

(1)在程序运行时，由于本题数据较多，运行时间较长，若有大量的数据可能不便于处理。

(2)当数据较少时，用 Kruskal 算法可能比较复杂。

5.2 模型改进

(1)当数据较少生成稀疏树时，采用 **Prim 算法**；当数据较多生成稠密树时，采用 **Kruskal 算法**。

具体原因如下：**Prim 算法**本质上以一个结点作为最小生成树的初始结点，然后以迭代的方式找出与最小生成树中各结点权重最小边，并加入到最小生成树中。加入之后如果产生回路则跳过这条边，选择下一个结点。这样一次次遍历使得 **Prim 算法**的时间复杂度为 $O(n^2)$，在处理较稠密的树时会花费较长的时间；相比之下 **Kruskal 算法**更适合处理稠密树，其本质为每次寻找图中可用的权值最小的边相连，若两端点不在同一棵树下则将其合并，这样由多棵小树合并成一个大树，少去了一次次遍历的过程，**Kruskal 算法**的时间复杂度为 O(nlogn)。(n 为边的条数)[6]

(2)由于模型需要计算的信息量、次数比较多，通过多种算法的结合希望可以有所简化。

参考文献

[1] 汪遐昌，《最小生成树问题》，四川师范大学学报(自然科学版)，1997.01

[2]司守奎等，《数学建模算法与应用》（第 2 版），北京：国防工业出版社，2016.1

[3]陈杰，《MATLAB 宝典》，电子工业出版社出版，2007

[4]陈健,多叉树的绘制算法研究及实现[J].福建商业高等专科学校学报,2008.06

[5]牛庆威,彭晓钰,丁林花,薛琳,并查集在程序竞赛中的应用[J].电脑编程技巧与维护,2018.09

[6]Martine Labbé,Miguel A. Pozo,Justo Puerto. Computational comparisons of different formulations for the Stackelberg minimum spanning tree game[J]. International Transactions in Operational Research,2021,28(1).

附录:

附录一
（Dec-C++编写的求解问题一的代码）

E_1.cpp:

```
#include<bits/stdc++.h>
using namespace std;
double dis[200][200];
struct point
{
    double x,y;
};
point po[200];
int main()
{
    freopen("求两点间的距离.txt","w",stdout);                    //重定向输出
    int n=180;
    double x,y;
    for(int i=1;i<=n;++i)
{
        cin>>po[i].x>>po[i].y;
}
    for(int i=1;i<=n;++i)
{
        for(int j=i;j<=n;++j)
        {
dis[i][j]=dis[j][i]=sqrt((po[i].x-po[j].x)*(po[i].x-po[j].x)+(po[i].y-po[j].y)*(po[i].y-
po[j].y));                                          //计算两点间的欧式距离
        }
    }
    for(int i=1;i<=n;++i)
{
        for(int j=i+1;j<=n;++j)
{
            cout<<i<<" "<<j<<" "<<dis[i][j]<<endl;       //输出点间的距离
        }
    }
    return 0;
}
```

E_2.cpp:

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
```

```
int n,m;
struct st
{
    int xx,yy,level;
    double v;
}
a[200005];
int fa[5005];
int find(int s)
{
    if(fa[s]!=s) fa[s]=find(fa[s]);
    return fa[s];
}
struct rule
{
    bool operator()(const st&s1,const st&s2)
     {
        if(s1.level!=s2.level) return s1.level<s2.level;
        return s1.v<s2.v;
     }
};
void hebing(int s1,int s2)
  {
    fa[s2]=s1;
}
int main()
{
    freopen("road.out","w",stdout);
    cin>>n>>m;                              //n 是点数，m 是边数，中心点为 0
    for(int i=1; i<=m; ++i)
    {
        cin>>a[i].xx>>a[i].yy>>a[i].v;
        if(a[i].xx<=12&&a[i].yy<=12) a[i].level=1;
        else a[i].level=2;
    }
    for(int i=1; i<=n; ++i)
   {
        fa[i]=i;
    }
    sort(a+1,a+1+m,rule());
    double tot=0;
    int k=0;
    cout<<"**********************************"<<endl;
    for(int i=1; i<=m; ++i)
```

```
    {
        int s1=find(a[i].xx),s2=find(a[i].yy);
        if(s1!=s2)
        {
            hebing(s1,s2);
            cout<<a[i].xx<<","<<a[i].yy<<","<<a[i].v<<endl; //输出哪两个点相连和
其间的距离
            tot+=a[i].v;
            k++;
        }
        if(k==n-1)break;
    }
    cout<<tot;
    return 0;
}
```

E_5.c:

```
#include <stdio.h>
#include <stdlib.h>
struct point
{
    int count;
    int num;
    double len;
}a[200],t;
void insert(int k);
void sort(int k);
int main()
{
    int n,i;
    FILE* head;
    scanf("%d",&n);
    insert(n);
    sort(n);
    head=freopen("二级供水站筛选结果.csv","w",stdout);
    for(i=1;i<n;i++)
    {
        if(a[i].count==1)
        printf("P%d,%lf\n",a[i].num-12,a[i].len);
    }
    return 0;
}
void insert(int k)
{
```

```
    int i,x,y;
     double z;
     for(i=1;i<=k;i++)
     a[i].count=0;
    for(i=1;i<=k;i++)
    {
        scanf("%d,%d,%lf",&x,&y,&z);
        a[x].num=x;
        a[y].num=y;
        a[x].count++;
        a[y].count++;
        a[x].len=z;
        a[y].len=z;
     }
}
void sort(int k)
{
    int i,j;
    for(i=1;i<k;i++)
    {
        for(j=i+1;j<=k;j++)
        {
            if(a[i].len<a[j].len)
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
        }
    }
}
```

E_6.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int c[12]={0};
double dist[12]={0.};
int num=0;
struct point
{
    double dis[10];
```

```
    int front;
    int son[10];
    int level;
}a[181];
struct data
{
    int x;
    int y;
    double len;
    int use;//在创建分支的时候一个数据只能用一次，否则多分支情况会卡住，停留在第一个分支上
}b[181];
struct da
{
    int x;
    int y;
}e[181];
void insert(int k);

int buildbranch(int k);

void bound(int k);

void find(int k);

int findson(int k);

void output(int k);

int order1(int k);

double length(int k);

void distance();

void out();

void length_of_PtoV(int k);

int llength(int k);

int main()
{
    int n,bon,i;
```

```
    scanf("%d",&n);//输入共有多少个节点
    insert(n-1); //  输入 180 个边的数据
    scanf("%d",&bon);//输入端点的个数
    bound(bon);//输入端点的信息
    find(n);
    for(i=0;i<10;i++)
     {
        if(c[i]==0)
        break;
        buildbranch(c[i]);
    }
    output(n);
    distance();
    out();
    //length_of_PtoV(n);
    llength(n);
}

void insert(int k)
{
    int i,x1,y1;
    double len1;
    for(i=0;i<k;i++)
    {
        scanf("%d,%d,%lf",&x1,&y1,&len1);
        b[i].x=x1;
        b[i].y=y1;
        b[i].len=len1;

    }
    for(i=0;i<k;i++)
    {
        b[i].use=1;
    }
    for(i=1;i<=k;i++) //设置各节点的等级
    {
        if(i<=12)
        a[i].level=1;
        else
        a[i].level=2;
    }

}
```

```
void bound(int k)
{
    int i,x1;
    double d;//数据里有一个距离但是这里没用，先用 d 存一下
    for(i=1;i<=k;i++)
    {
        scanf("%d,%lf",&x1,&d);
        a[x1].son[0]=0;
    }
}



int buildbranch(int k)
{
    int i=0;
    findson(k);
    for(i=0;i<=6;i++)
    {
        if(a[k].son[i]==0)
        return 0;
        buildbranch(a[k].son[i]);
    }
}

int findson(int k)
{
    int i,j,u,t;
    t=k;


        for(i=0;i<=180;i++)
        {
            if(b[i].x==t&&b[i].use)
           {
            j=0;
            a[t].son[j]=b[i].y;//存储分支
            a[t].dis[j++]=b[i].len;//存储对应的距离
            b[i].use=0;//该数据已被使用
            a[b[i].y].front=t;//设置下一个节点的父节点
            u=i;
            while(u<=180)
            {
                u++;
```

```
            if(b[u].x==t&&b[u].use)
            {

                a[t].son[j]=b[u].y;//存储分支
                  a[t].dis[j++]=b[u].len;//存储对应的距离
                  b[u].use=0;//该数据已被使用
                  a[b[u].y].front=t;//设置下一个节点的父节点
            }
            else if(b[u].y==t&&b[u].use)
            {

                a[t].son[j]=b[u].x;
                a[t].dis[j++]=b[u].len;
                b[u].use=0;
                a[b[u].x].front=t;
            }
        }

      }
else if(b[i].y==t&&b[i].use)
{
      j=0;
      a[t].son[j]=b[i].x;//存储分支
      a[t].dis[j++]=b[i].len;//存储对应的距离
      b[i].use=0;//该数据已被使用
      a[b[i].x].front=t;//设置下一个节点的父节点
      u=i;
      while(u<=180)
      {
            u++;
            if(b[u].x==t&&b[u].use)
            {

                a[t].son[j]=b[u].y;//存储分支
                  a[t].dis[j++]=b[u].len;//存储对应的距离
                  b[u].use=0;//该数据已被使用
                  a[b[u].y].front=t;//设置下一个节点的父节点
            }
            else if(b[u].y==t&&b[u].use)
            {

                a[t].son[j]=b[u].x;
                a[t].dis[j++]=b[u].len;
                b[u].use=0;
```

```
                    a[b[u].x].front=t;
                }
            }
        }
        }

}

void find(int k)
{
    int i,count=0;
    for(i=0;i<=k;i++)
    {
        if(b[i].x==0)
        {
            c[count++]=b[i].y;

        }
        else if(b[i].y==0)
        {
            c[count++]=b[i].x;

        }

    }
    findson(0);
}

void output(int k)
{
    int i,j;
    for(i=0;i<k;i++)
    {
        if(i>12)
        printf("P%d 的子节点为:,\n",i-12);
        else if(i>0)
        printf("V%d 的子节点为:,\n",i);
        else
        printf("%d 的子节点为:,\n",i);
        for(j=0;j<10;j++)
        {
            if(a[i].son[j]==0)
            break;
            if(a[i].son[j]>12)
```

```
                printf(",P%d,%lf\n",a[i].son[j]-12,a[i].dis[j]);
                else if(a[i].son[j]>0)
                printf(",V%d,%lf\n",a[i].son[j],a[i].dis[j]);

            }
            printf("\n");
        }
}

double length(int k)
{
    int i=0;
    double ll=0.;
    if(a[k].son[0]==0)
    return ll;
    else
    {
         for(i=0;i<10;i++)
         {

            if(a[k].son[i]==0)
            break;
            ll+=a[k].dis[i];
            if(a[a[k].son[i]].level==1)
            continue;
            ll+=length(a[k].son[i]);

        }
        return ll;
    }
}

void distance()
{
    int i;
    for(i=0;i<12;i++)
    {
        dist[i]=length(i+1);
    }
}
void out()
{
    int i;
    for(i=0;i<12;i++)
```

```
    printf("V%d 的总里程为:,%lf\n",i+1,dist[i]);
}

int llength(int k)
{
    int i,j,x1,y1;
    for(i=0;i<k;i++)
    {
        scanf("%d,%d",&x1,&y1);
        e[i].x=x1;
        e[i].y=y1;
    }
         printf(",A,V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12\n");
    for(i=13;i<k;i++)
    {
        j=1;
        printf("P%d:,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf\n",i-
12,sqrt(pow((e[i].x-e[0].x),2)+pow((e[i].y-e[0].y),2)),sqrt(pow((e[i].x-
e[1].x),2)+pow((e[i].y-e[1].y),2)),sqrt(pow((e[i].x-e[2].x),2)+pow((e[i].y-
e[2].y),2)),sqrt(pow((e[i].x-e[3].x),2)+pow((e[i].y-e[3].y),2)),sqrt(pow((e[i].x-
e[4].x),2)+pow((e[i].y-e[4].y),2)),sqrt(pow((e[i].x-e[5].x),2)+pow((e[i].y-
e[5].y),2)),sqrt(pow((e[i].x-e[6].x),2)+pow((e[i].y-e[6].y),2)),sqrt(pow((e[i].x-
e[7].x),2)+pow((e[i].y-e[7].y),2)),sqrt(pow((e[i].x-e[8].x),2)+pow((e[i].y-
e[8].y),2)),sqrt(pow((e[i].x-e[9].x),2)+pow((e[i].y-e[9].y),2)),sqrt(pow((e[i].x-
e[10].x),2)+pow((e[i].y-e[10].y),2)),sqrt(pow((e[i].x-e[11].x),2)+pow((e[i].y-
e[11].y),2)),sqrt(pow((e[i].x-e[12].x),2)+pow((e[i].y-e[12].y),2)));
    }
    return 0;
}
```

附录二

（MATLAB 编写的求解问题一、二、三的代码）

E_3.m:

```
filename1='数据集.csv';
filename2='二级供水站.csv';
DATA=csvread(filename1,0,2);
LINE1=csvread(filename2);

% 绘制出各点
scatter(DATA(1,1),DATA(1,2),'O');
hold on
for i=2:13
     scatter(DATA(i,1),DATA(i,2),'r*')
```

```
    hold on
end
for i=14:181
    scatter(DATA(i,1),DATA(i,2),'b.')
    hold on
end

% 计算两种型号管道总长度
sumA=0;
sumB=0;

% 绘制出连线
for i=1:180
    if  (LINE1(i,1) <= 12) && (LINE1(i,2) <= 12)

plot([DATA(LINE1(i,1)+1,1),DATA(LINE1(i,2)+1,1)],[DATA(LINE1(i,1)+1,2),DATA(
LINE1(i,2)+1,2)],'Color','r');
        hold on
        sumA=sumA+LINE1(i,3);
    else

plot([DATA(LINE1(i,1)+1,1),DATA(LINE1(i,2)+1,1)],[DATA(LINE1(i,1)+1,2),DATA(
LINE1(i,2)+1,2)],'Color','g');
        hold on
        sumB=sumB+LINE1(i,3);
    end
end
```

E_4.m:

```
filename1='数据集.csv';
filename2='二级供水站.csv';
DATA=csvread(filename1,0,2);
LINE1=csvread(filename2);

% 绘制出各点
scatter(DATA(1,1),DATA(1,2),'O');
hold on
text(DATA(1,1),DATA(1,2),'A');
for i=2:13
    scatter(DATA(i,1),DATA(i,2),'r*')
    hold on
    t=num2str(i-1);
    text(DATA(i,1),DATA(i,2),strcat('V',t),'FontSize',8);
end
```

```
for i=14:181
    scatter(DATA(i,1),DATA(i,2),'b.')
    hold on
    t=num2str(i-13);
    text(DATA(i,1),DATA(i,2),strcat('P',t),'Color','red','FontSize',6);
end

% 绘制出连线
for i=1:180
    if   (LINE1(i,1) <= 12) && (LINE1(i,2) <= 12)

plot([DATA(LINE1(i,1)+1,1),DATA(LINE1(i,2)+1,1)],[DATA(LINE1(i,1)+1,2),DATA(
LINE1(i,2)+1,2)],'Color','k');
        hold on
    else

plot([DATA(LINE1(i,1)+1,1),DATA(LINE1(i,2)+1,1)],[DATA(LINE1(i,1)+1,2),DATA(
LINE1(i,2)+1,2)],'Color','g');
        hold on
    end
end
```