

**数学建模论文**

**题目： A题自来水管道铺设问题**

**队伍成员： 张舒航**

**陈乐乐**

**魏江涛**

**目录**

[**摘要** 1](#_Toc25555)

[**一、问题的重述** 2](#_Toc689)

[1.1问题背景 2](#_Toc1973)

[1.2问题提出 2](#_Toc20736)

[**二、问题假设** 2](#_Toc13449)

[**三、符号说明** 3](#_Toc12897)

[**四、问题一的模型建立与求解** 5](#_Toc19313)

[**五、问题二的模型建立与求解** 7](#_Toc25765)

[5.1问题分析 7](#_Toc22350)

[5.2算法描述 8](#_Toc16952)

[5.3实验结果 9](#_Toc32700)

[**六、问题三的模型建立与求解** 10](#_Toc14236)

[6.1问题分析 11](#_Toc15652)

[6.2解决流程 12](#_Toc27072)

[6.3染色与剪枝 **错误!未定义书签。**](#_Toc7745)

[6.4实验结果 13](#_Toc15345)

[**七、模型的评价** 14](#_Toc12988)

[7.1优点 14](#_Toc6777)

[7.2缺点 14](#_Toc3030)

[**参考文献** 1](#_Toc17579)3

[**附录** 1](#_Toc17579)4

**摘要**

现代日常生活中需要通过自来水管道将自来水运输至各个用户处，本文主要分析讨论自来水自来水管道连接规划问题，使自来水管道将各个供水点用最短路劲链接，以达到节约成本，实现资源有效利用的目的。

对于问题一，要求铺设总里程数最少，即是分层的最小生成树问题。采用prim算法来构建模型，我们先分析一级水站到二级水站的情况，我们拿所有的一级水站作为初始点去构造最小生成树，基于这个强假设，第一层和第二层的最小生成树是不会互相影响的，所以二层的最小生成树（一型供水站到二型供水站）和一层的最小生成树（A水站到一型水站）分别最小能保证总的路程最小。

对于问题二，要求替换两个二型供水站，使Ⅱ型管道里程数最少，我们先运用了两个最小生成树进行拼接，再对生成带环图进行分析，利用水站不需要同时输入水，我们就利用以去环为原则升级二型供水站，为保证Ⅱ型管道最短则优先消除最长Ⅱ型管道来消除环。对于出现断环的情况，我们运用深度优先搜索的基本思想。

对于问题三，我们对提出的假设，在问题一的基础上，将无向图转化为以中心水站为起点的有向图，从中心水站开始进行深度优先搜索并染色，循环进行直到无法进行染色，取出未染色的无水数根结点视为待处理结点，将其周围的最近的有水结点进行升级，清空染色后再次染色。

关键词：管道连接；最小生成树；带环图；prim算法；染色

### 一 问题重述

#### 1.1问题背景

现代日常生活中，需要通过自来水管道将自来水运输至各个用户处，本文主要分析讨论自来水管道连接规划问题，即在自来水管道铺设过程使管道的里程数最少。

#### 1.2问题信息与基本条件

在村村通自来水工程实施过程中，从保证供水质量以及设备维护方便角度出发，某地区需要建设一个中心供水站，12个一级供水站和168个二级供水站，各级供水站的位置坐标如附录2所示，其中类型A表示中心供水站，类型V代表一级供水站，类型P为二级供水站。

现在要将中心供水站A处的自来水通过管道输送到一级供水站和二级供水站。按照设计要求，从中心站A铺设到一级供水站的管道为I型管道，从一级供水站出发铺设到二级供水站的管道为II型管道。

自来水管道铺设技术要求如下：

1. 中心供水站只能和一级供水站连接（铺设I型管道），不能和二级供水站直接相连，但一级供水站之间可以连接（铺设I型管道）。
2. 一级供水站可以与二级供水站相连（铺设II型管道），且二级供水站之间也可以连接（铺设II型管道）。
3. 各级供水站之间的连接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，不能从任意管道中间的一点进行连接。
4. 相邻两个供水站之间（如果有管道相连）所需管道长度可简化为欧氏距离。

#### 1.3问题重述

问题1：从中心供水站A出发，自来水管道应该如何铺设才能使管道的总里程最少？以图形给出铺设方案，并给出I型管道和II型管道总里程数。

问题2：由于II型管道市场供应不足，急需减少从一级供水站出发铺设的II型管道总里程，初步方案是将其中两个二级供水站升级为一级供水站。问选取哪两个二级供水站，自来水管道应该如何铺设才能使铺设的II型管道总里程最少？相对问题1的方案，II型管道的总里程减少了多少公里？

问题3：在问题1基础上，假如现实中由于功率的影响，从一级供水站出发铺设的管道最多只能供水40公里（按从该一级供水站管道输送的总里程计算），但从中心供水站A出发铺设的管道供水不受此距离限制。为实现对所有供水站供水，需要将若干个二级供水站升级为一级供水站，但升级后从该供水站出发铺设的管道也最多只能供水40公里。问最少升级几个二级供水站，可实现对所有的供水站供水？在这种配置下铺设管道的总里程数最少是多少公里？

### 二 符号说明

|  |  |
| --- | --- |
| 符号 | 定义 |
|  | 无向图 |
|  | 无向图的第个结点, |
|  | 无向图的结点与结点相连的边, |
|  | 中心水站集合 |
|  | 一级水站集合(为了避免歧义没有使用) |
|  | 二级水站集合 |
|  | 无向图结点总数， |
|  | 无向图边的总数， |
|  | 结点的坐标 |
|  | 结点的坐标 |
|  | 的某个子集 |
|  | 的某个子集 |
|  | 树， |
|  | 包含结点集合 |
|  | 图中的环， |

### 三 数据预处理

#### 3.1数据标识

题目给出的数据存在“序号”的“类型”两个唯一的字段来作为识别数据的方法。但因为题目中出现升级水站的操作，为了避免歧义，我们用序号来对水站进行唯一的标识：表示序号为的水站。

#### 3.2数据表示与存储

通过分析我们很容易可以明确，应该使用无向图来表示数据，而对于本题构造的无向图应满足：

显然，对于本题我们理应用邻接矩阵来储存无向图的数据：因为这是一个全连通图，使用邻接矩阵来储存可以占有更少的空间；并且由于内存分布连续，对缓存也更加友好。无论时间与空间性能都较优。

但在这里，我们仍然选用邻接表来表示图。

原因是邻接表可以在的时间复杂度内获取任意结点的所有邻居，而邻接矩阵则需要。考虑到之后需要的遍历操作，以及对无用边的删除，所以我们选用邻接表进行数据的储存。图3-1给出了邻接表的通常示意图。

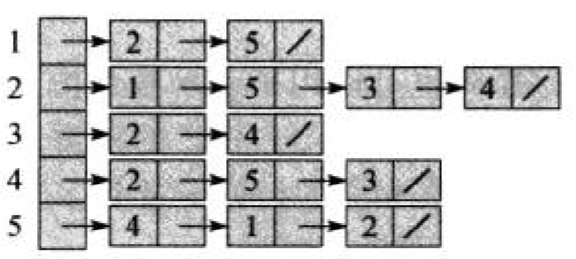


图3-1

### 四 问题一的分析建模与求解

#### 4.1问题分析

根据题意，我们需要构建一个包含所有结点并且权重和最小的连通图，即寻找满足以下条件的和。

那么便很容易想到通过构建最小生成树来达成这个目的。但是值得注意的是，每个结点的相接条件并不一致，一级供水站与二级供水站之间可以两两任意连接，而中心供水站却只能与一级供水站相接。由此我们可以先构筑出一棵包含所有结点与结点的最小生成树，最后再加入结点，并将其与最邻近的结点相连。

#### 4.2算法描述

我们选用Prim算法来构造最小生成树。Prim算法是一种基于贪心策略构建最小生成树的一种算法，以下给出Prim算法的最基本通用流程：

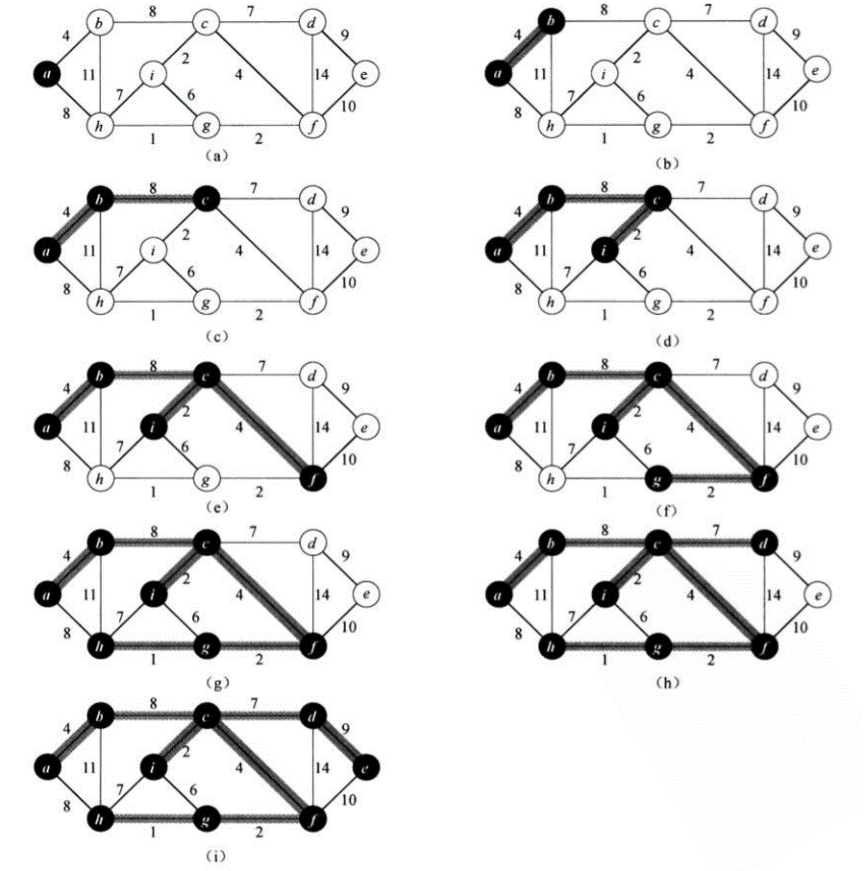


图4-1

1).输入：一个带权连通图，其中顶点集合为，边集合为；  
2).初始化：，其中,称为起始结点；  
3).重复下列操作，直到：

a.在中选取权值最小的边，其中满足：

b.将加入，将加入中；

4).输出：最小生成树 ，满足。

#### 4.3 模型优势

构建最小生成树的算法有很多，除了本文给出的Prim算法之外，还有Kruskal算法和Boruvka算法等等，但本文依旧选用了Prim算法，原因如下：

1).实现简单，同时有扩展优化的空间，可以利用优先队列进行改进并将时间复杂度优化到；

2).在使用邻接表存储的情况下可以在遍历时在的时间复杂度下删除已存的边，对后续的遍历更为友好。

#### 4.4问题解答

管道总长为451.68，其中I型管道总长11.66，II型管道总长440.02。（结果保留两位小数），图4-2为仿真结果。

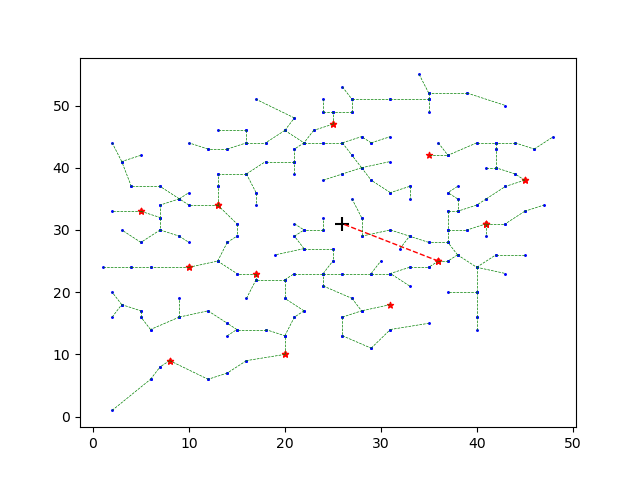


图4-2 红线为I型管道，绿线为II型管道.

### 五、问题二的分析建模与求解

#### 5.1问题分析

#### 5.1.1准备工作

**定理1**：**最优方案总是无环图。**

**证明：**假设最优方案可能不是无环图。那么对于这个方案总能找到一个环。那么环中存在一个结点，根据环的定义，存在一条从出发并回归自身的路径，在这个路径上总能找到一个与相连的结点，那么就存在着两条不同的从出发到达的路径，删除边得到的依旧是连通图，而，说明不是最优的，与假设矛盾。

**定理2：要使一个环断开，最多只能删除一条边。**

**证明：**根据环的定义，存在一条从出发并回归自身的路径，那么删除这条路径的一条边，这个路径所构成的环断开。

#### 5.1.2分析过程

为了尽可能的减少II型管道的长度，我们优先使用I型管道。

首先对于与分别生成一棵最小生成树，然后将两棵树接合成一个新的无向图：

其中

不一定是一个无环图，而根据定理1，最优方案一定是无向图，那么我们就需要在图中断环。

根据题意，当我们找到了环之后，我们应该优先断开II型管道，那么根据定理2，我们仅能断开一条边，为了II型管道消耗最少，我们选择环中最长的一条II型管道来断开。

而这时候引入升级水站的操作，我们需要尽可能地借此断开较长的II型管道，那么我们可以把最长的II型管道包括在环中，进行断环，我们自然就可以把它断开了，重复这样的操作两次即可。

#### 5.2算法描述

#### 5.2.1深度优先搜索的概述

我们需要一个算法来寻找图中的环，此处我们使用带访问标记的深度优先搜索算法，它的大致流程为：

1).图中某个顶点出发出发，首先访问，并记录。

2).然后访问该顶点的邻居，此时可能出现两种情况：

a.访问到了某个访问过的结点了，说明找到了环，可以直接退出并返回当前结点与之间的所有结点。

b.访问的是一个全新的结点，那么继续递归，如果它已经没有没访问过的邻居了，便可以返回。

#### 5.2.3回溯(backtrack)和深度优先搜索的组合使用

想象一个场景，当我们找到了一个环，知道了它的首尾结点，我们如何才能快速获得这个环内的所有结点信息？

如果我们再遍历一次，就花费了多余的时间，因为这个环越大，再次遍历的时间就越多；而如果我们一边搜索一边记录，最后的记录里则会出现很多不在环内的结点。

因此，我们引入回溯算法，以图5-1为例，进行搜索。

考虑维护一个栈来记录路径，并将搜索的记录都压入栈中。

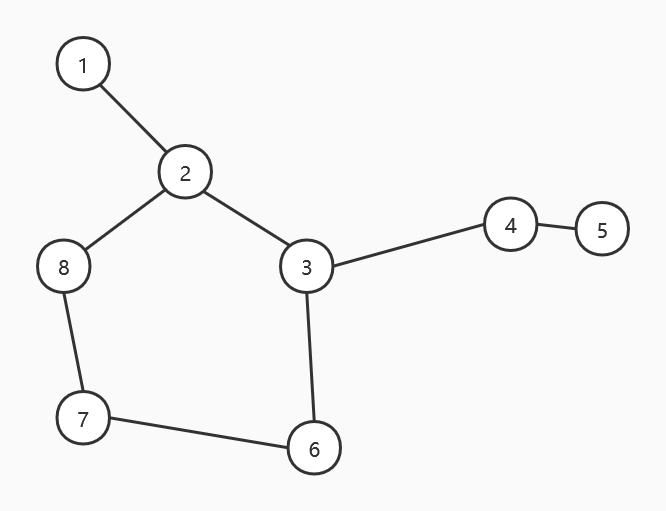
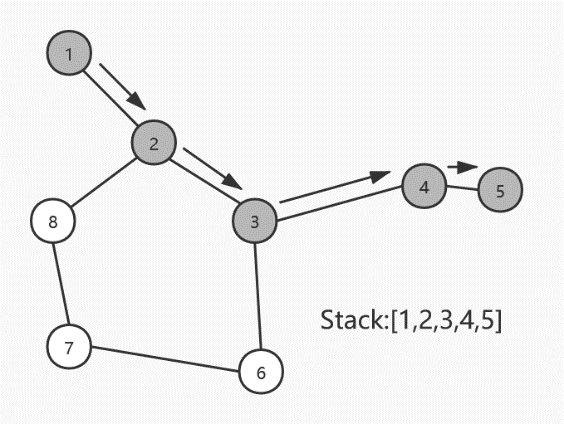
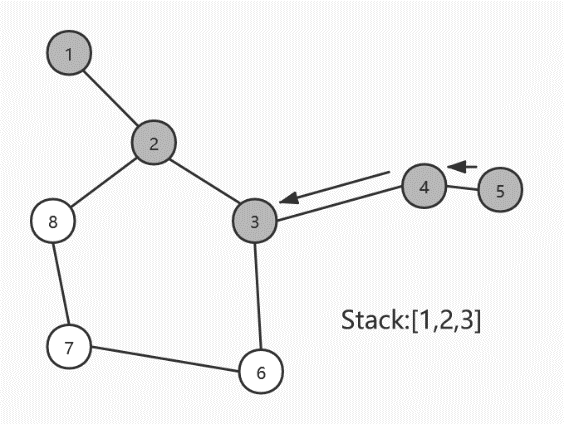


图5-1

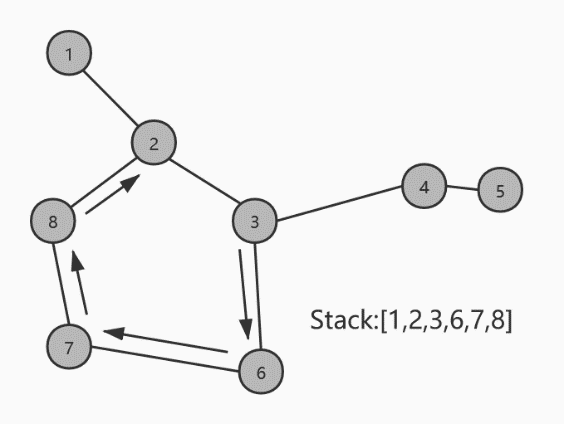
先搜索到终止条件，并没有发现环：



那么开始回溯，同时一边把记录出栈：



继续搜索另一分支，这时便找到了环，而栈尾的正是我们要找的环。



#### 5.3模型优势

广度优先搜索同样是常用的图搜索算法，本文选用深度优先搜索，原因如下：

1).广度优先搜索需要使用队列作为辅助数据结构，虽然深度优先搜索同样需要用栈来辅助递归，但这并不需要在程序中写出，而是由操作系统分配，方便程序的编写。

2).深度优先搜索与回溯算法契合度更高，理念上更为接近。

#### 5.4 问题解答

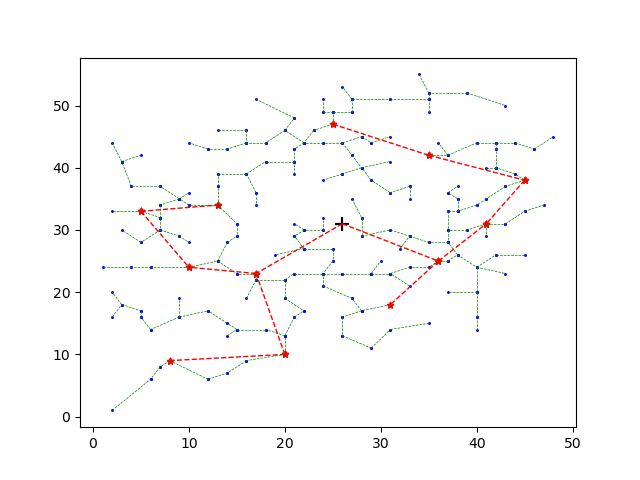


图5-2未断环

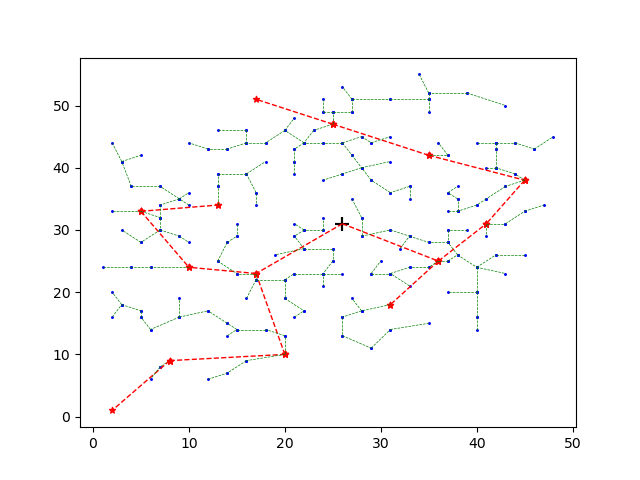


图5-3 问题2结果

最终解答如图5-3所示，图5-2为未断环前的示意图。

需要升级位于 (2, 1) 处的 125 号水站。

需要升级位于 (17, 51) 处的 89 号水站。

管道总长为532.12，其中I型管道总长139.89，II型管道总长392.24。（结果保留两位小数）

与问题1的方案进行比较，II型管道总长大约减少了47.78。

### 六、问题三的分析建模与求解

#### 6.1问题分析

#### 6.1.1问题假设

基于题意，我们定义一个概念——流速，用表示，它具有以下几个性质。

1).每个结点都有流速，用表示，且有；但当时，此点的流速无意义，我们称结点是无水的。

2).当结点有水时，结点的流速总是足够大，而结点的流速固定为40。

3).结点一开始总是无水的，但水可以在结点之间传播。水的传播是有方向的，水总会从流速高的结点流向流速低的结点，除非流速低的结点为流速高的结点的父结点。一般认为水会从结点开始传播。

4).一般的，对于类结点存在个流速大于它的结点时，有：

5).特别的，当类结点周围仅存在1个流速大于它的结点时，有：

#### 6.1.2分析过程

对题干提出的功率概念进行解读可知：中心水站和一级水站会将收到的水用新的功率泵出，无论上一个水站的功率有多大；而二级水站只是让水流过；水流在没有重新泵出的情况下最多只能经过40km。

基于此我们在6.1.1提出了关于流速的假设，本质是在解释每个水站的最大送水能力。在此题里流量的概念并没有办法很好的契合题目，尤其是在合流的时候，应该选用速度最大的支流作为当前的流速。

而下文将全盘基于以上概念进行分析；且根据题意，将在问题1的方案上进行分析。

问题1的方案为一棵最小生成树，即是一个连通无环图。虽然根据假设，我们理应改用有向图来描述本小题的模型，但起码在这个环节里不需要转化，因为图里没有环，也不会存在多个点的水输入某一个结点的情况。

我们先从开始对图进行染色，计算每一点的流速，我们便可以得到一批无水结点。

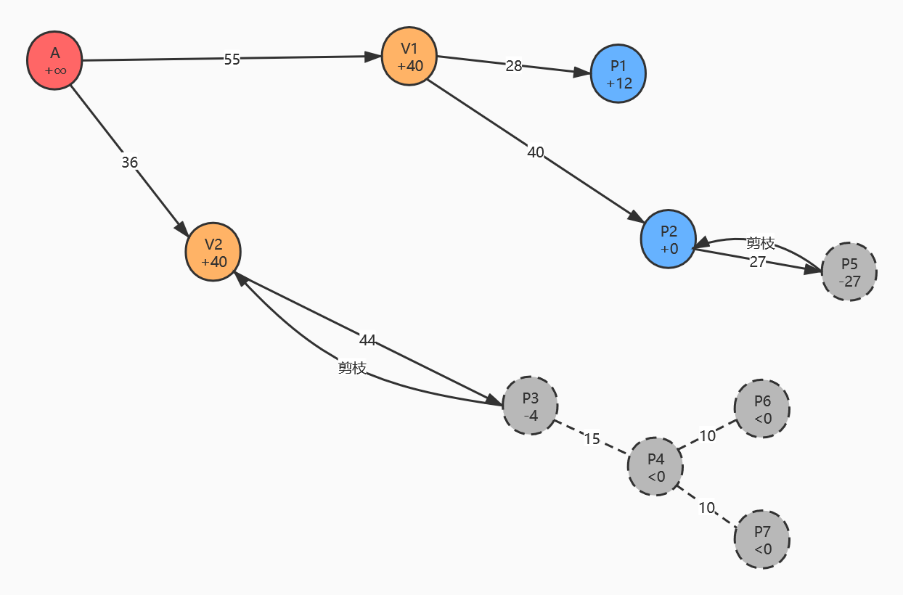
得到了无水结点之后，我们便要考虑如何升级结点，值得注意的是，被升级的结点并不一定是无水结点。如果升级的是无水结点，那么需要在升级后需要与结点（或是距其最近40km以内的结点）相连，需要多花费I型管道，但也可以根据水的回流来断开II型管道（转化为有向图）；而如果升级的是有水结点，那么便不需要增设新的管道。

在不确定升级策略的情况下，本文采用贪心算法作为寻找升级结点的策略。

#### 6.2模型建立

#### 6.2.1染色与剪枝

首先我们需要对图中的每一点计算出相应的流速值，我们仍旧沿用深度优先搜索的方法，只需要在遍历的时候为结点赋上父结点的流速和边的距离的差即可。不过在此处我们可以通过定义发现：在遍历的过程中，如果结点是无水的，那么的所有子孙结点都是无水的，这就会出现一棵无水子树，即为无水子树的根结点，那么我们便无需对子树进行搜索了，这就完成了一次剪枝。



#### 6.2.2贪心算法

我们已经找到了个无水结点，这时如果升级结点，会减少个无水结点，那么我们需要找到满足

的和，那么此时剩余无水结点数变为

当时，继续如上操作，知道为止。

接着计算相应方案的管道里程数。

#### 6.3模型优势

若能对贪心算法进行严格证明，那么这个算法能拥有较好的鲁棒性，对于各种类型的数据都可以得出正确结论。

剪枝操作操作负担低，效率高。

#### 6.4实验结果

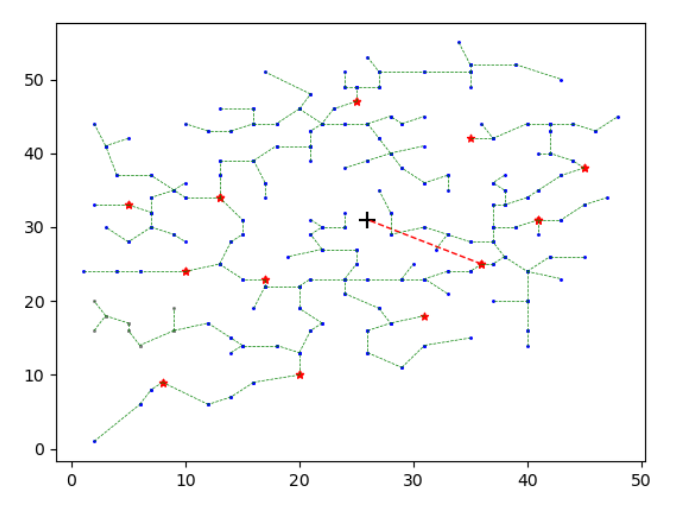


图6-1 染色结果

图6-1为深度优先染色后的结果，右边的灰色结点即为无水结点，既然只有一棵无水子树，那么便可以不使用贪心算法，直接升级距离无水树根结点最近的有水结点即可

升级1个结点，管道长度和问题1方案一致，管道总长均为为451.68。

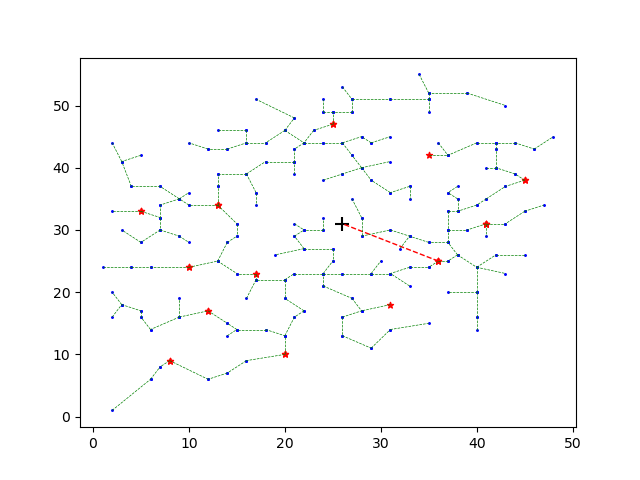


图6-2 问题3仿真结果

### 七、模型的评价与调优

#### 7.1 对于问题1模型的讨论

#### 7.1.1复杂度

算法的时间复杂度取决于最小优先队列的实现方式，若使用二叉堆时时间复杂度为，使用斐波那契堆实现时时间复杂度将改进到。

邻接表的空间复杂度约为。

#### 7.1.2评价

最小生成树的生成可以一步到位，不需要进行人工的修改。

#### 7.2对于问题2模型的讨论

#### 7.2.1复杂度

递归回溯拥有最好情况的时间复杂度，其中为环的数量。

#### 7.2.2评价

回溯算法运用空间换时间，拥有较深度优先搜索更好的复杂度。但实际上在搜索环的问题上还有其他效果不错的算法，例如CACM 491[[1]](#footnote-1)算法或是其余的诸多算法[[2]](#footnote-2)，这些算法可以高效的一次性获得图中所有的环，避免深度优先搜索带来的重复遍历。

#### 7.3对于问题3模型的讨论

#### 7.3.1复杂度

因为剪枝操作的存在，深度优先搜索可以在的时间内完成染色。

#### 7.3.2评价

问题三中模型所求得的解是针对本题数据的最优解，却未必是整个问题的最优解，如果需要进一步推广，应该证明贪心算法在这个问题上是全局最优的。另外还需要证明升级有水结点要比升级无水结点更合理，而且根据不同的条件与标准得出的结论是不同的，这里仅以升级水站个数作为标准，如果加入对里程数的计算并适当赋予其一定的权值得出的结论应该会更符合实际、更利于推广。

### 参考文献

[1]司守奎，孙兆亮编著.数学建模算法与应用 [M].国防工业出版社:北京.

[2]Thomas H.Cormen，Charles E.Leiserson，Ronald L.Rivest，Clifford Stein.算法导论（原书第3版）[M].机械工业出版社:北京,2012-12:341.

[3]NetworkX.Official NetworkX source code repository.[EB/OL].https://github.com/networkx/networkx,2020-6.

### 附录

#### 附录1解题代码

'''

运行环境：Python 3.6.6 [MSC v.1900 64 bit (AMD64)] on win32

第三方依赖：

    matplotlib         3.2.1

    networkx           2.4

    numpy              1.18.4

    pandas             1.0.1

'''

# dataform.py

# 封装文件，封装了A题所需要的数据和图论操作

import matplotlib.pyplot as plt

import networkx as nx

import numpy as np

import pandas as pd

def euclidean\_distance(p1, p2):

    x1, y1, x2, y2 = \*p1[-2:], \*p2[-2:]

return np.sqrt((x1-x2)\*\*2+(y1-y2)\*\*2)

class DataForm:

    ''' 数据表封装类

    封装从csv文件中读取的水站数据，并可以对其进行一定的操作。

    @Attributes:

        df{pandas.DataFrame} : 存储的数据表

    '''

    def \_\_init\_\_(self, path: str):

        self.df = pd.read\_csv(path, encoding='utf-8')

    def Get(self, i) -> pd.DataFrame:

        return self.df.loc[i]

    def GetByType(self, types: str) -> pd.DataFrame:

        return self.df[self.df['TYPE'].str.contains(types)]

    def GetType(self, i: int) -> str:

        return self.df.loc[i, 'TYPE'][0]

    def CheckRoute(self, i: int, j: int) -> bool:

        p1\_type, p2\_type = self.Get(i)['TYPE'], self.Get(j)['TYPE']

        return (('V' in p1\_type) or ('A' in p1\_type)) and (('V' in p2\_type) or ('A' in p2\_type))

    def GetPlotList(self, i: int, j: int) -> tuple:

        p1 = self.Get(i)

        p2 = self.Get(j)

        return (p1['X'], p2['X']), (p1['Y'], p2['Y'])

    def BuildGraph(self, typeSelecter='A|V|P', dis=euclidean\_distance) -> nx.Graph:

        graph = nx.Graph()

        form = self.GetByType(typeSelecter).to\_numpy()

        for i in form:

            for j in form:

                if i[0] != j[0]:

                    graph.add\_edge(i[0], j[0], weight=dis(i, j))

        return graph

    def Draw(self):

        A, V, P = self.GetByType('A'), self.GetByType('V'), self.GetByType('P')

        plt.scatter(A['X'], A['Y'], marker='+', c='black', s=100)

        plt.scatter(V['X'], V['Y'], marker='\*', c='red', s=20)

        plt.scatter(P['X'], P['Y'], marker='.', c='blue', s=5)

    def Upgrade(self, i, target='V'):

        self.df.loc[i, 'TYPE'] = target

    def WriteToFile(self, path: str):

        self.df.to\_csv(path)

    def BestNeighbor(self, base: int, types='A|V', minimum=True):

        neighbors = self.BuildGraph(types)[base]

        if minimum:

            weight = np.Infinity

        else:

            weight = 0

        for key in neighbors:

            if minimum:

                if neighbors[key]['weight'] < weight:

                    weight = neighbors[key]['weight']

                    target = key

            else:

                if neighbors[key]['weight'] > weight:

                    weight = neighbors[key]['weight']

                    target = key

        return (base, target, weight)

def graphDraw(dataform: DataForm, graph: nx.Graph):

    def drawPoint(i):

        p = dataform.Get(i)

        ty = dataform.GetType(i)

        if 'A' in ty:

            marker, c, s = '+', 'black', 100

        elif 'V' in ty:

            marker, c, s = '\*', 'red', 20

        elif 'P' in ty:

            marker, c, s = '.', 'blue', 5

        try:

            if graph.nodes[i]['power'] < 0:

                c = 'gray'

        except:

            pass

        plt.scatter(p[-2], p[-1], marker=marker, c=c, s=s)

    for i, j, dis in graph.edges.data('weight'):

        if dataform.CheckRoute(i, j):

            color = 'red'

            lw = 1

        else:

            color = 'green'

            lw = 0.5

        plotList = dataform.GetPlotList(i, j)

        plt.plot(\*plotList, linewidth=lw, c=color, linestyle='--')

        drawPoint(i)

        drawPoint(j)

def merge(lhs: nx.Graph, rhs: nx.Graph) -> nx.Graph:

    graph = nx.Graph()

    for i, j, dis in lhs.edges.data('weight'):

        graph.add\_edge(i, j, weight=dis)

    for i, j, dis in rhs.edges.data('weight'):

        graph.add\_edge(i, j, weight=dis)

    return graph

def weightStats(dataform: DataForm, graph: nx.Graph):

    route\_1, route\_2 = 0, 0

    for i, j, dis in graph.edges.data('weight'):

        if dataform.CheckRoute(i, j):

            route\_1 += dis

        else:

            route\_2 += dis

    return route\_1+route\_2, route\_1, route\_2

def maxEdge(dataform: DataForm, graph: nx.Graph, grade=False):

    res = (0, 0, 0)

    for i, j, w in graph.edges.data('weight'):

        if dataform.CheckRoute(i, j) is grade:

            if w > res[-1]:

                res = i, j, w

    return res

def disconnectCycle(dataform: DataForm, graph: nx.Graph, cycle: list, grade=False, minimum=False):

    if len(cycle[0]) is 2:

        cycleList = cycle

    else:

        cycleList = np.array(np.array(cycle)[:, :2], dtype=int)

    if minimum:

        weight = np.Infinity

    else:

        weight = 0

    for i, j in cycleList:

        if grade is None or dataform.CheckRoute(i, j) is grade:

            if minimum:

                if graph.edges[i, j]['weight'] < weight:

                    weight = graph.edges[i, j]['weight']

                    res = (i, j)

            else:

                if graph.edges[i, j]['weight'] > weight:

                    weight = graph.edges[i, j]['weight']

                    res = (i, j)

    graph.remove\_edge(\*res)

    return res

# solve1.py

# 问题1的解

import matplotlib.pyplot as plt

import networkx as nx

import numpy as np

import pandas as pd

import dataform as dfm

if \_\_name\_\_ == "\_\_main\_\_":

    center = 0

    df = dfm.DataForm(".\\code\\anotherA\\data.csv")

    # 构筑一级水站和二级水站间的最小生成树。

    G = nx.minimum\_spanning\_tree(df.BuildGraph('V|P'), algorithm='prim')

    # 筛选出距离中心水站最近的一级水站，并加入图中。

    for\_add = df.BestNeighbor(center)

    G.add\_edge(\*for\_add[:2], weight=for\_add[2])

    # 输出

    dfm.graphDraw(df, G)

    print("管道总长为%.2fkm，其中I型管道总长%.2fkm，II型管道总长%.2fkm。（结果保留两位小数）"

          % dfm.weightStats(df, G))

    plt.savefig('.\\code\\anotherA\\result\\solve1.png')

    plt.show()

# solve2.py

# 问题2的解

import matplotlib.pyplot as plt

import networkx as nx

import numpy as np

import pandas as pd

import dataform as dfm

def buildUncycleGraph(df, G):

    # 每次深度优先搜索时寻找一个环，并将环中最长的II型管道断开，直到图中没有环为止。

    while True:

        try:

            dfm.disconnectCycle(df, G, nx.find\_cycle(G))

        except:

            break

    return G

def dfs(df, G, edge):

    # 以 i 为起点向 j 方向做深度优先搜索，直到尽头或是搜索到一级水站。

    i, j = edge[:2]

    # 终止条件1：如果搜索结果全部都是图尽头，说明i比j更接近高级水站，那么就升级j。

    if len(G[j]) <= 1:

        return True

    # 终止条件2：如果搜索到了一个高级水站，说明j距离高级水站更近，所以应该升级i。

    type\_j = df.GetType(j)

    if type\_j is 'A' or type\_j is 'V':

        return False

    # 递归dfs。

    for key in G[j]:

        if key is not i:

            if not dfs(df, G, (j, key)):

                return False

    return True

def smartUpgrade(df, G, count):

    # 需要记录升级的结点作为返回值

    uplist = []

    for \_ in range(count):

        i, j = dfm.maxEdge(df, G)[:2]

        # 理论上升级结点构成的环只需要包含最长边似乎就行了，那就直接取最长边的边沿来升级。

        if dfs(df, G, (i, j)):

            df.Upgrade(j)

            upgraded = j

        else:

            df.Upgrade(i)

            upgraded = i

        uplist.append(upgraded)

        # 因为有结点升级了，所以需要添加新的I型管道。

        for\_add = df.BestNeighbor(upgraded)

        G.add\_edge(\*for\_add[:2], weight=for\_add[2])

        # 添加了新管道图中就有环了，那么直接移除最长管道来断环，不需要再搜索了。

        G.remove\_edge(i, j)

    return uplist

if \_\_name\_\_ == "\_\_main\_\_":

    upgradeCount = 2

    df = dfm.DataForm(".\\code\\anotherA\\data.csv")

    # 将两个最小生成树接合成一个连通图。

    G = dfm.merge(nx.minimum\_spanning\_tree(df.BuildGraph('A|V'), algorithm='prim'),

                  nx.minimum\_spanning\_tree(df.BuildGraph('V|P'), algorithm='prim'))

    dfm.graphDraw(df, G)

    plt.savefig('.\\code\\anotherA\\result\\solve2\_base.png')

    plt.clf()

    # dfs自动断环。

    buildUncycleGraph(df, G)

    # 升级2个处于最长边周围，并可以使图中出现环的结点，并断开这些环。

    uplist = smartUpgrade(df, G, upgradeCount)

    # 输出

    for i in uplist:

        msg = df.Get(i)

        print("需要升级位于 (%d, %d) 处的 %d 号水站。" % (msg['X'], msg['Y'], i))

    print("管道总长为%.2fkm，其中I型管道总长%.2fkm，II型管道总长%.2fkm。（结果保留两位小数）"

          % dfm.weightStats(df, G))

    dfm.graphDraw(df, G)

    plt.savefig('.\\code\\anotherA\\result\\solve2.png')

    plt.show()

# solve3.py

# 问题3的解

import matplotlib.pyplot as plt

import networkx as nx

import numpy as np

import pandas as pd

import dataform as dfm

def fillWater(df, G, driedRoots=[], begin=None, end=0, power=np.Infinity):

    # 深度优先搜索，同时对结点染色，如果水流经一级结点还>=0的话，就重置为40，小于0时说明到头了，记录结点后返回。

    if power < 0:

        driedRoots.append(end)

        return

    if 'V' in df.GetType(end):

        power = 40.0

    G.nodes[end]['power'] = power

    for key in G[end]:

        if key is not begin:

            fillWater(df, G, driedRoots, end, key,

                      power - G[end][key]['weight'])

if \_\_name\_\_ == "\_\_main\_\_":

    center = 0

    df = dfm.DataForm(".\\code\\anotherA\\data.csv")

    G = nx.minimum\_spanning\_tree(df.BuildGraph('V|P'), algorithm='prim')

    for\_add = df.BestNeighbor(center)

    G.add\_edge(\*for\_add[:2], weight=for\_add[2])

    # 初始化结点权重。

    for i in G.nodes:

        G.nodes[i]['power'] = -1

    # 染色

    driedRoots = []

    fillWater(df, G, driedRoots)

    dfm.graphDraw(df, G)

    plt.savefig('.\\code\\anotherA\\result\\solve3\_distance\_compute.png')

    plt.clf()

    # 将无水树的根结点最近的有水结点升级。

    node = driedRoots[0]

    for i in G[node]:

        if G.nodes[i]['power'] >= 0:

            df.Upgrade(i)

            break

    msg = df.Get(node)

    print("需要升级位于 (%d, %d) 处的 %d 号水站。" % (msg['X'], msg['Y'], i))

    # 再染一次色。

    fillWater(df, G)

    dfm.graphDraw(df, G)

    plt.savefig('.\\code\\anotherA\\result\\solve3.png')

    plt.show()

#### 附录2 预处理数据

ID,TYPE,X,Y

0,A,26,31

1,V1,5,33

2,V2,8,9

3,V3,10,24

4,V4,13,34

5,V5,17,23

6,V6,20,10

7,V7,25,47

8,V8,31,18

9,V9,35,42

10,V10,36,25

11,V11,41,31

12,V12,45,38

13,P1,41,35

14,P2,40,34

15,P3,38,35

16,P4,38,37

17,P5,33,37

18,P6,31,36

19,P7,33,35

20,P8,28,32

21,P9,24,30

22,P10,21,31

23,P11,22,27

24,P12,28,29

25,P13,43,37

26,P14,44,39

27,P15,25,27

28,P16,21,29

29,P17,22,30

30,P18,24,32

31,P19,37,33

32,P20,38,33

33,P21,37,36

34,P22,14,13

35,P23,16,9

36,P24,14,7

37,P25,18,14

38,P26,12,6

39,P27,15,14

40,P28,20,13

41,P29,13,46

42,P30,16,39

43,P31,21,39

44,P32,26,44

45,P33,28,40

46,P34,27,42

47,P35,29,38

48,P36,29,44

49,P37,36,44

50,P38,41,40

51,P39,39,52

52,P40,27,49

53,P41,23,46

54,P42,20,46

55,P43,16,46

56,P44,22,44

57,P45,40,44

58,P46,42,40

59,P47,37,42

60,P48,35,49

61,P49,35,51

62,P50,35,52

63,P51,34,55

64,P52,26,53

65,P53,27,51

66,P54,31,51

67,P55,31,45

68,P56,31,41

69,P57,28,45

70,P58,27,35

71,P59,24,38

72,P60,26,39

73,P61,13,37

74,P62,17,36

75,P63,21,41

76,P64,18,41

77,P65,21,43

78,P66,13,39

79,P67,14,43

80,P68,12,43

81,P69,10,44

82,P70,16,44

83,P71,18,44

84,P72,24,44

85,P73,25,49

86,P74,24,49

87,P75,24,51

88,P76,21,48

89,P77,17,51

90,P78,10,34

91,P79,9,35

92,P80,7,37

93,P81,4,37

94,P82,5,42

95,P83,2,44

96,P84,7,32

97,P85,7,30

98,P86,1,24

99,P87,2,16

100,P88,3,18

101,P89,2,20

102,P90,4,24

103,P91,5,28

104,P92,6,24

105,P93,9,29

106,P94,2,33

107,P95,7,34

108,P96,3,30

109,P97,3,41

110,P98,10,36

111,P99,17,34

112,P100,20,22

113,P101,24,21

114,P102,22,17

115,P103,21,16

116,P104,27,19

117,P105,26,16

118,P106,9,16

119,P107,12,17

120,P108,14,15

121,P109,19,26

122,P110,14,28

123,P111,13,25

124,P112,9,19

125,P113,2,1

126,P114,6,6

127,P115,7,8

128,P116,6,14

129,P117,5,17

130,P118,5,16

131,P119,16,19

132,P120,26,13

133,P121,29,11

134,P122,31,14

135,P123,28,17

136,P124,20,19

137,P125,17,22

138,P126,15,23

139,P127,21,23

140,P128,24,23

141,P129,26,23

142,P130,25,25

143,P131,15,31

144,P132,15,29

145,P133,10,28

146,P134,38,26

147,P135,37,25

148,P136,33,21

149,P137,40,24

150,P138,44,44

151,P139,41,31

152,P140,33,24

153,P141,32,27

154,P142,40,14

155,P143,42,26

156,P144,45,33

157,P145,29,23

158,P146,31,30

159,P147,30,25

160,P148,31,23

161,P149,35,15

162,P150,40,16

163,P151,40,20

164,P152,37,20

165,P153,35,24

166,P154,43,23

167,P155,45,26

168,P156,37,28

169,P157,35,28

170,P158,33,29

171,P159,37,30

172,P160,39,30

173,P161,41,29

174,P162,43,31

175,P163,47,34

176,P164,46,43

177,P165,42,43

178,P166,48,45

179,P167,42,44

180,P168,43,50

1. Paton, K. An algorithm for finding a fundamental set of cycles of a graph. Comm. ACM 12, 9 (Sept 1969), 514-518. [↑](#footnote-ref-1)
2. Finding all the elementary circuits of a directed graph. D. B. Johnson, SIAM Journal on Computing 4, no. 1, 77-84, 1975. https://doi.org/10.1137/0204007

   Enumerating the cycles of a digraph: a new preprocessing strategy. G. Loizou and P. Thanish, Information Sciences, v. 27, 163-182, 1982.

   A search strategy for the elementary cycles of a directed graph. J.L. Szwarcfiter and P.E. Lauer, BIT NUMERICAL MATHEMATICS, v. 16, no. 2, 192-204, 1976. [↑](#footnote-ref-2)