

# 数据结构上机实验报告

庄震丰 22920182204393

Jun. 9<sup>th</sup>, 2020

## 1 实现聊天电脑版微信

### 1.1 需求分析

实现一个简单的聊天功能的程序。

### 1.2 实现过程

一个简单的局域网多人聊天室软件。实现局域网内的聊天室创建、搜索和加入。必须在局域网的条件下使用 (暂不完全支持热点, 打开热点的用户将无法使用)。使用时要在设置中打开软件的通知权限。

创建聊天室: 创建服务器并加入到自己的聊天室中, 同时向局域网可见。

结构分为: Client 端和 server 端。

实现语言: python

主要功能 (用户手册):

- 运行服务器建立连接 server.py
- 运行客户端进行登陆 client.py
- 名称更改 setname nickname
- 信息发送 send message
- 寻求帮助 help [命令名称]
- ...

### 1.3 实现代码

clinet.py

```
1  import socket
2  import threading
3  import json
4  from cmd import Cmd
5  import datetime
6
7  class Client(Cmd):
8      """客户端
9
10     """
11     prompt = ''
12     intro = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+ '[Welcome] 简易聊天室\n'+ '[Welcome] 输入来获取帮助 help\n'
13
14     def __init__(self):
15         """构造
```

```
16
17     """
18     super().__init__()
19     self.__socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20     self.__id = None
21     self.__nickname = None
22
23     def __receive_message_thread(self):
24         """接受消息线程
25
26         """
27         while True:
28             # noinspection PyBroadException
29             try:
30                 buffer = self.__socket.recv(1024).decode()
31                 obj = json.loads(buffer)
32                 print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+[' ' + str
                    (obj['sender_nickname']) + '(' + str(obj['sender_id']) + ')' + ''], obj
                        ['message'])
33             except Exception:
34                 print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+['Client]
                    无法从服务器获取数
                    据'])
35
36     def __send_message_thread(self, message):
37         """发送消息线程
38
39         :param message: 消息内容
40         """
41         self.__socket.send(json.dumps({
42             'type': 'broadcast',
43             'sender_id': self.__id,
44             'message': message
45         }).encode())
46
47     def start(self):
48         """启动客户端
49
50         """
51         self.__socket.connect(('127.0.0.1', 8888))
52         self.cmdloop()
53
54     def do_login(self, args):
55         """登录聊天室
56
57         :param args: 参数
58         """
59         nickname = args.split(' ')[0]
60
61         # 将昵称发送给服务器, 获取用户id
62         self.__socket.send(json.dumps({
63             'type': 'login',
```

```

64         'nickname': nickname
65     }).encode())
66     # 尝试接受数据
67     # noinspection PyBroadException
68     try:
69         buffer = self.__socket.recv(1024).decode()
70         obj = json.loads(buffer)
71         if obj['id']:
72             self.__nickname = nickname
73             self.__id = obj['id']
74             print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[ Client ]
              成功登录到聊天
              室')
75
76         # 开启子线程用于接受数据
77         thread = threading.Thread(target=self.__receive_message_thread)
78         thread.setDaemon(True)
79         thread.start()
80     else:
81         print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[ Client ]
              无法登录到聊天
              室')
82     except Exception:
83         print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[ Client ] 无法
              从服务器获取数据')
84
85     def do_send(self, args):
86         """发送消息
87
88         :param args: 参数
89         """
90         message = args
91         # 显示自己发送的消息
92         print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+[' + str(self.
              __nickname) + '(' + str(self.__id) + ')' + ']', message)
93         # 开启子线程用于发送数据
94         thread = threading.Thread(target=self.__send_message_thread, args=(message, ))
95         thread.setDaemon(True)
96         thread.start()
97
98     def do_help(self, arg):
99         """帮助
100
101         :param arg: 参数
102         """
103         command = arg.split(' ')[0]
104         if command == '':
105             print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n')
106             print('[Help] login nickname - 登录到聊天室, 是你选择的昵称nickname')
107             print('[Help] send message - 发送消息, 是你输入的消息message')
108         elif command == 'login':
109             print('[Help] login nickname - 登录到聊天室, 是你选择的昵称nickname')

```

```
110         elif command == 'send':
111             print('[Help] send message - 发送消息, 是你输入的消息message')
112         else:
113             print('[Help] 没有查询到你想要了解的指令')
114     client = Client()
115     client.start()
```

#### server.py

```
1     import socket
2     import threading
3     import json
4     import datetime
5
6     class Server:
7         """服务器类
8
9         """
10        def __init__(self):
11            """构造
12
13            """
14            self.__socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15            self.__connections = list()
16            self.__nicknames = list()
17
18        def __user_thread(self, user_id):
19            """用户子线程
20
21            :param user_id: 用户id
22            """
23            connection = self.__connections[user_id]
24            nickname = self.__nicknames[user_id]
25            print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[Server] 用
                户', user_id, nickname, '加入聊天室')
26            self.__broadcast(message='用户 ' + str(nickname) + '(' + str(user_id) + ')' + '加入聊天
                室')
27
28        # 侦听
29        while True:
30            # noinspection PyBroadException
31            try:
32                buffer = connection.recv(1024).decode()
33                # 解析成数据json
34                obj = json.loads(buffer)
35                # 如果是广播指令
36                if obj['type'] == 'broadcast':
37                    self.__broadcast(obj['sender_id'], obj['message'])
38            else:
39                print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[Server]
                    无法解析数据
                    包json:', connection.getsockname(), connection.fileno())
```

```
40         except Exception:
41             print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[Server] 连接
               失效:', connection.getsockname(), connection.fileno())
42             self.__connections[user_id].close()
43             self.__connections[user_id] = None
44             self.__nicknames[user_id] = None
45
46 def __broadcast(self, user_id=0, message=''):
47     """广播
48
49     :param user_id: 用户id为系统(0)
50     :param message: 广播内容
51     """
52     for i in range(1, len(self.__connections)):
53         if user_id != i:
54             self.__connections[i].send(json.dumps({
55                 'sender_id': user_id,
56                 'sender_nickname': self.__nicknames[user_id],
57                 'message': message
58             }).encode())
59
60 def start(self):
61     """启动服务器
62
63     """
64     # 绑定端口
65     self.__socket.bind(('127.0.0.1', 8888))
66     # 启用监听
67     self.__socket.listen(10)
68     print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[Server] 服务器正在运
               行.....')
69
70     # 清空连接
71     self.__connections.clear()
72     self.__nicknames.clear()
73     self.__connections.append(None)
74     self.__nicknames.append('System')
75
76     # 开始侦听
77     while True:
78         connection, address = self.__socket.accept()
79         print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+'[Server] 收到一个
               新连接', connection.getsockname(), connection.fileno())
80
81         # 尝试接受数据
82         # noinspection PyBroadException
83         try:
84             buffer = connection.recv(1024).decode()
85             # 解析成数据json
86             obj = json.loads(buffer)
87             # 如果是连接指令, 那么则返回一个新的用户编号, 接收用户连接
88             if obj['type'] == 'login':
```

```

89         self.__connections.append(connection)
90         self.__nicknames.append(obj['nickname'])
91         connection.send(json.dumps({
92             'id': len(self.__connections) - 1
93         })).encode())
94
95         # 开辟一个新的线程
96         thread = threading.Thread(target=self.__user_thread, args=(len(self.
97             __connections) - 1, ))
98         thread.setDaemon(True)
99         thread.start()
100     else:
101         print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+[Server]
102             无法解析数据
103             包json:', connection.getsockname(), connection.fileno())
104 except Exception:
105     print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+'\n'+[Server] 无法
106         接受数据:', connection.getsockname(), connection.fileno())
107
108 server = Server()
109 server.start()

```

## 1.4 实现结果

The image shows two side-by-side screenshots of a Windows command prompt window titled 'C:\Windows\py.exe'. The left window shows the server's output for a user named 'zzf'. It starts with a welcome message, followed by a login attempt, successful login, and then a message being sent. The right window shows the server's output for a user named 'dawnbreaker'. It also starts with a welcome message, followed by a login attempt, successful login, and then a message being sent. Both windows show the server's internal state, including the number of connections and the list of nicknames.

```

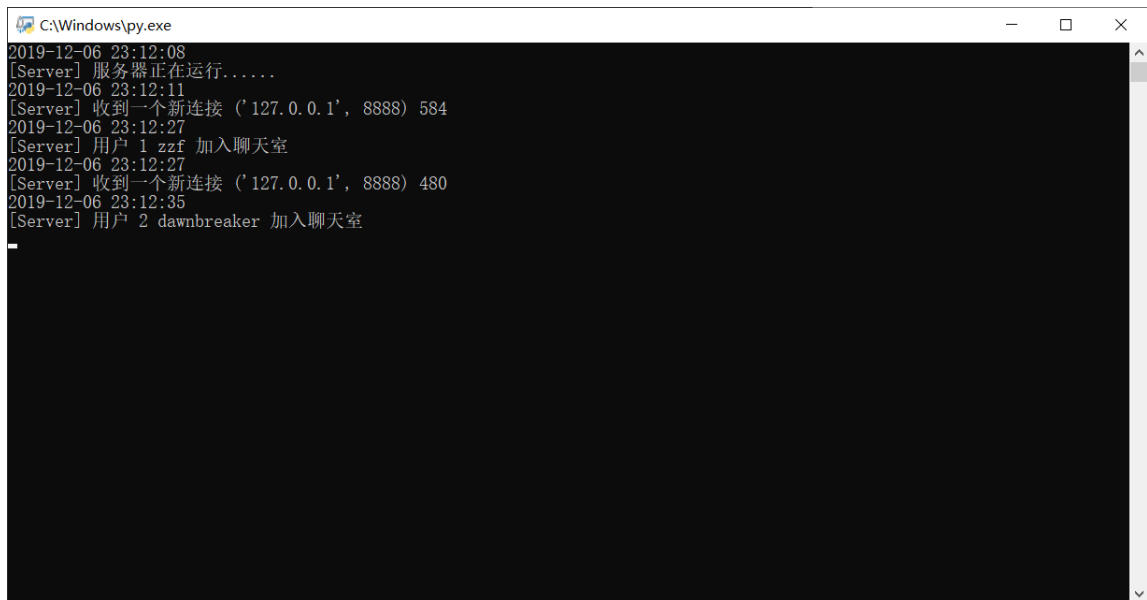
C:\Windows\py.exe
2019-12-06 23:12:11
[Welcome] 简易聊天室客户端
[Welcome] 输入help来获取帮助

login zzf
2019-12-06 23:12:27
[Client] 成功登录到聊天室
2019-12-06 23:12:27
[System(0)] 用户' zzf(1)加入聊天室
2019-12-06 23:12:35
[System(0)] 用户' dawnbreaker(2)加入聊天室
2019-12-06 23:12:40
[dawnbreaker(2)] hello!
help login
[Help] login nickname - 登录到聊天室, nickname是你选择的昵称
send I'm zzf
2019-12-06 23:13:22
[zzf(1)] I'm zzf

C:\Windows\py.exe
2019-12-06 23:12:14
[Welcome] 简易聊天室客户端
[Welcome] 输入help来获取帮助

help
2019-12-06 23:12:24
[Help] login nickname - 登录到聊天室, nickname是你选择的昵称
[Help] send message - 发送消息, message是你输入的消息
login dawnbreaker
2019-12-06 23:12:35
[Client] 成功登录到聊天室
2019-12-06 23:12:35
[System(0)] 用户' dawnbreaker(2)加入聊天室
send hello!
2019-12-06 23:12:40
[dawnbreaker(2)] hello!
help send
[Help] send message - 发送消息, message是你输入的消息
2019-12-06 23:13:22
[zzf(1)] I'm zzf

```



```

C:\Windows\py.exe
2019-12-06 23:12:08
[Server] 服务器正在运行.....
2019-12-06 23:12:11
[Server] 收到一个新连接 ('127.0.0.1', 8888) 584
2019-12-06 23:12:27
[Server] 用户 1 zzf 加入聊天室
2019-12-06 23:12:27
[Server] 收到一个新连接 ('127.0.0.1', 8888) 480
2019-12-06 23:12:35
[Server] 用户 2 dawnbreaker 加入聊天室

```

图一：项目一的结果截图

## 1.5 优化总结

- 可以对窗口式的命令行进行渲染，以达到简洁易上手的 App 应用的目的
- 可以利用 python 的画图功能将用户进行显示
- 可以将 python 的程序应用进行打包处理，达到扩展应用的目的

## 2 朋友圈关系

### 2.1 实现目的

利用图论解决朋友圈关系的紧密程度的查询问题。

### 2.2 实现过程

利用朋友圈两两之间的关系建立点和边，针对不同的关系给予不同的边权值。紧密程度定义为：两点间的平均路径权与最短路之间的层次函数，该函数定义为，对于此两个不同的参考因子取不同的比例，所能得到的最大值，作为两点间的关系。

即：

$$f(v_i, v_j) = \max_j (M_{\frac{n(n-1)}{2} \times 2} \cdot P_{2 \times n})$$

其中 M 是：

$$\begin{bmatrix} \sum w_{u_1 v_1} & dis[i_1][j_1] \\ \sum w_{u_2 v_2} & dis[i_2][j_2] \\ \dots & \dots \end{bmatrix}$$

P 代表不同的权重取值，但是， $p[i][1] + p[i][2] = 1$

对于不同的关系，给出了不同的亲密程度值（越小代表越亲密）

关系类型	mother	father	son	dauther	boyfriend	girlfriend	wife
权值	4	4	3	3	8	8	5
husband	classmate	stranger	teacher	friend	boss	employee	student
5	15	30	16	12	23	23	16

关系类型到权值的映射采用了 hash 函数进行对应，键值为字符串，返回类型为关系对应权值。  
并且会返回朋友圈最紧密的关系路线

## 2.3 实现代码

```
1 #include<bits/stdc++.h>
2 #define maxn 1000
3 const int INF=0x3f3f3f3f;
4 using namespace std;
5 int n,m,q,ans=0,pth=INF,route=0;
6 int G[maxn][maxn];
7 bool vis[maxn][maxn];
8 int vrel[100];
9 bool rec[maxn][maxn];
10 string H[maxn];
11 void dfs(int x,int y)
12 {
13     if (x==y)
14     {
15         route=route+ans;
16         if (pth>ans)
17         {
18             pth=min(pth,ans);
19             for (int i=1;i<=n;i++)
20                 for (int j=1;j<=n;j++)
21                     rec[i][j]=vis[i][j];
22         }
23         return;
24     }
25     for (int i=1;i<=n;i++)
26         if (G[x][i]!=INF&&!vis[x][i])
27         {
28             vis[x][i]=true;
29             vis[i][x]=true;
30             ans=ans+G[x][i];
31             dfs(i,y);
32             ans=ans-G[x][i];
33             vis[x][i]=false;
34             vis[i][x]=false;
35         }
36 }
37 int Hash(string a)
38 {
39     int key=0;
40     for (int i=0;i<a.length();i++)
41         key=key+a[i];
42     key=key%100;
43     while (H[key]!="") key=(key+1)%100;
44     H[key]=a;
45     return key;
46 }
47 int Index(string a)
```



```
48 {
49     int k=0;
50     for (int i=0;i<a.length();i++)
51         k=k+a[i];
52     k=k%100;
53     while(H[k]!=a) k++;
54     return k;
55 }
56 void printmap(int x)
57 {
58     cout<<x;
59     for (int i=1;i<=n;i++)
60         if (rec[x][i])
61         {
62             cout<<"->";
63             rec[x][i]=0;
64             rec[i][x]=0;
65             printmap(i);
66             break;
67         }
68 }
69 void init()
70 {
71     vrel[55]=4;
72     vrel[34]=4;
73     vrel[36]=3;
74     vrel[49]=3;
75     vrel[62]=8;
76     vrel[63]=8;
77     vrel[27]=5;
78     vrel[41]=5;
79     vrel[57]=15;
80     vrel[70]=30;
81     vrel[32]=16;
82     vrel[33]=12;
83     vrel[39]=23;
84     vrel[64]=23;
85     vrel[70]=16;
86     for (int i=0;i<maxn;i++)
87         for (int j=0;j<maxn;j++)
88         {
89             vis[i][j]=false;
90             G[i][j]=INF;
91         }
92     cout<<"please input the vertex and edge number:";
93     cin>>n>>m;
94     for (int i=1;i<=m;i++)
95     {
96         int pre,aft;
97         string v;
98         cout<<"please add <v1,v2,relate>:";
99         scanf("%d%d",&pre,&aft);
```

```

100     getline(cin,v);
101     //cout<<Index(v);
102     G[pre][aft]=vrel[Index(v)];
103     G[aft][pre]=vrel[Index(v)];
104 }
105 cout<<"please input the query times:";
106 cin>>q;
107 for (int i=1;i<=q;i++)
108 {
109     int s,e;
110     cout<<"start vertex and terminated vertex:";
111     cin>>s>>e;
112     ans=0;
113     route=0;
114     pth=INF;
115     memset(vis,false,sizeof(vis));
116     dfs(s,e);
117     if (route)
118     {
119         cout<<"friend distance:"<<route<<endl;
120         cout<<"strongest path:"<<pth<<endl;
121         printmap(s);
122     }
123     else cout<<"No relationship!"<<endl;
124 }
125 }
126
127 int main()
128 {
129     string a[15]={"mother","father","son","dauther","boyfriend","girlfriend","wife","husband",
130                 "classmate","stranger","teacher","friend","boss","employee","student"};
131     for (int i=0;i<15;i++)
132         Hash(a[i]);
133     init();
134     return 0;
135 }

```

## 2.4 设计和调试分析

- please input the vertex and edge number 输入点和边的数目
- please add <vi,vj,relate> 输入一条边对应的关联点和关系类型
- please input the query times 输入查询次数
- start vertex and terminated vertex 输入起始点和终点

## 2.5 测试结果

```
please input the vertex and edge number:5 6
please add <v1,v2,relate>:1 2 father
please add <v1,v2,relate>:1 3 boss
please add <v1,v2,relate>:2 3 classmate
please add <v1,v2,relate>:2 4 teacher
please add <v1,v2,relate>:3 5 wife
please add <v1,v2,relate>:4 5 stranger
please input the query times:2
start vertex and terminated vertex:1 4
friend distance:158
strongest path:20
1->2->4start vertex and terminated vertex:2 5
friend distance:232
strongest path:20
2->3->5
```

图二：朋友圈关系测试结果

如图：friend distance 代表 (1,4) 这一对点的边权和，最优路径权是 20，给出的最优路径是 1-2-4  
对于 (2,5)，边权是和是 232，最优路径权 20，最优路径为 2-3-5

### 3 动态最短路

#### 3.1 需求分析

当有边权发生变化时，给出较优化的算法去基于原来的 dij 函数去快速求出新的询问下的最短路。

#### 3.2 概要设计

首先容易想到改变一个边权之后重新跑一边 dijkstra 算法，这样当有  $q$  次改变时对应时间复杂度达到  $O(qn^2)$ 。为了减少计算时间，采用的链路变化情况分为两部分，权重增大和减小。

当权重增大时，受用集合  $\text{sub}(E(e))$  的节点和这些节点相关的链路才会发生变化，其他的节点链路不会发送任何变化。当权重减小时， $\text{sub}(E(e))$  的节点的最短路径减少为  $d=w(e)-w'(e)$ ，如果节点的最短路径值发生改变，则重新选择最短路一定会经过  $e$ 。假设  $N$  为节点集合，包含所有需要更新的节点，当  $N$  的节点有一个  $M.v.\text{inc}$  代表  $v$  节点的所有入边重最小增量值的集合， $L$  为链路集合，存储所有变化的链路，每个元素  $e$  有一个  $\text{min-inc}$  属性，代表节点  $E(e)$  最短路径的增加值，且可为负值，当有链路需要加入集合  $L$  时，执行入队操作  $L$ ，出队操作将选择  $L$  中  $\text{min}$  最小的链路，并将该链路从  $L$  中移除，利用线段树维护最小值即可。

算法复杂度分析：

假设有一个边的权值发生变化， $Q$  表示最短路径变化的节点的集合， $T_p$  是  $Q$  中节点的数目，参数  $Q$  和  $T_p$  只依赖网络拓扑结构和初始的 SPT， $D_{\max}$  表示和一个更新节点相连的链路数目的最大值。

复杂度为  $O(T_p^2 + T_p^2 D_{\max})$ ，考虑当链路的权重发生大的变化时，执行一次将花费  $O(T_p)$  的时间用于线性数组的查找，并且有  $T_p$  次这样的迭代，当一个父亲节点更新时，有个  $\text{sub}(v)$  次子节点的更新， $\text{sub}(v) < T_p$ ，父亲系欸但的更新迭代次数不超过  $T_p$ ，则有  $\sum_{v \in N} |\text{sub}(v)| = T_p$  假设一条链路的更新需要花费  $O(1)$  的时间，则所有的和更新节点关联的链路更新需要  $O(T_p^2 D_{\max})$ 。所以算法的运行总时间为  $O(T_p^2 + T_p^2 D_{\max})$ 。

#### 3.3 详细设计

```

1  #include <bits/stdc++.h>
2  #define pii pair<int, int>
3  #define LL long long
4  #define piii pair<pii, int>
5  #define ls(x) (x << 1)
6  #define rs(x) ((x << 1) | 1)
7  using namespace std;
8  const int maxn = 200010;
9  vector<piii> G[maxn];
10 piii edge[maxn];
11 int n;
12 map<int, int> mp;
13 void add(int x, int y, int z, int id) {
14     G[x].push_back(make_pair(make_pair(y, z), id));
15     G[y].push_back(make_pair(make_pair(x, z), id));
16 }
17 struct node {
18     bool flag;
19     LL mi, Set;
20 };
21 struct SegmentTree {
22     int n;
23     node tr[maxn * 4];

```

```
24     void pushup(int o) {
25         tr[o].mi = min(tr[ls(o)].mi, tr[rs(o)].mi);
26     }
27
28     void maintain(int o, LL val) {
29         tr[o].Set = min(tr[o].Set, val);
30         tr[o].mi = min(tr[o].mi, tr[o].Set);
31         tr[o].flag = 1;
32     }
33
34     void pushdown(int o) {
35         if(tr[o].flag) {
36             maintain(ls(o), tr[o].Set);
37             maintain(rs(o), tr[o].Set);
38             tr[o].Set = 1e18;
39             tr[o].flag = 0;
40         }
41     }
42
43     void build(int o, int l, int r) {
44         if(l == r) {
45             tr[o].mi = 1e18;
46             tr[o].flag = 0;
47             tr[o].Set = 1e18;
48             return;
49         }
50         int mid = (l + r) >> 1;
51         build(ls(o), l, mid);
52         build(rs(o), mid + 1, r);
53         tr[o].mi = 1e18;
54         tr[o].flag = 0;
55         tr[o].Set = 1e18;
56     }
57
58     void update(int o, int l, int r, int ql, int qr, LL val) {
59         if(ql > qr) return;
60         if(l == 0) return;
61         if(l >= ql && r <= qr) {
62             maintain(o, val);
63             return;
64         }
65         pushdown(o);
66         int mid = (l + r) >> 1;
67         if(ql <= mid) update(ls(o), l, mid, ql, qr, val);
68         if(qr > mid) update(rs(o), mid + 1, r, ql, qr, val);
69         pushup(o);
70     }
71
72     LL query(int o, int l, int r, int pos) {
73         if(l == r && l == pos) {
74             return tr[o].mi;
75         }
```

```

76         pushdown(o);
77         int mid = (l + r) >> 1;
78         if(pos <= mid) return query(ls(o), l, mid, pos);
79         else return query(rs(o), mid + 1, r, pos);
80     }
81 };
82 SegmentTree st;
83 struct Dj {
84     priority_queue<pair<long long, int>> q;
85     pii pre[maxn];
86     bool in_line[maxn], v[maxn], in_tree[maxn], is_line[maxn];
87     LL dis[maxn];
88     vector<int> G1[maxn];
89     int f[maxn];
90     vector<int> line;
91     vector<LL> re;
92
93     void add1(int x, int y) {
94         G1[x].push_back(y);
95         G1[y].push_back(x);
96     }
97
98     void dijkstra(int s) {
99         memset(v, 0, sizeof(v));
100        memset(dis, 0x3f, sizeof(dis));
101        q.push(make_pair(0, s));
102        dis[s] = 0;
103        while(q.size()) {
104            int x = q.top().second;
105            q.pop();
106            if(v[x]) continue;
107            v[x] = 1;
108            for (int i = 0; i < G[x].size(); i++) {
109                int y = G[x][i].first.first;
110                LL z = G[x][i].first.second;
111                if(v[y]) continue;
112                if(dis[y] > dis[x] + z) {
113                    dis[y] = dis[x] + z;
114                    pre[y] = make_pair(x, G[x][i].second);
115                    q.push(make_pair(-dis[y], y));
116                }
117            }
118        }
119    }
120
121    void dfs(int x, int flag, int fa) {
122        f[x] = flag;
123        for (int i = 0; i < G1[x].size(); i++) {
124            int y = G1[x][i];
125            if(y == fa || is_line[y]) continue;
126            dfs(y, flag, x);
127        }

```

```

128     }
129
130     void solve(int s) {
131         for (int i = 1; i <= n; i++) {
132             if (i == s) continue;
133             add1(i, pre[i].first);
134             in_tree[pre[i].second] = 1;
135         }
136         for (int i = n + 1 - s; i != s; i = pre[i].first) {
137             line.push_back(i);
138             in_line[pre[i].second] = 1;
139             is_line[i] = 1;
140         }
141         line.push_back(s);
142         is_line[s] = 1;
143         for (int i = 0; i < line.size(); i++) {
144             int y = line[i];
145             dfs(y, y, -1);
146         }
147     }
148 };
149 Dj dj1, dj2;
150 int main() {
151     int x, y, z, m, T;
152     cout<<"please input the points,edges,query numbers:";
153     scanf("%d%d%d", &n, &m, &T);
154     cout<<"input the order of the both points and edge value:";
155     for (int i = 1; i <= m; i++) {
156         scanf("%d%d%d", &x, &y, &z);
157         edge[i] = make_pair(make_pair(x, y), z);
158         add(x, y, z, i);
159     }
160     dj1.dijkstra(1), dj2.dijkstra(n);
161     dj1.solve(1), dj2.solve(n);
162     int cnt = 0;
163     for (int i = dj1.line.size() - 1; i >= 0; i--) {
164         mp[dj1.line[i]] = ++cnt;
165     }
166     st.build(1, 1, cnt - 1);
167     for (int i = 1; i <= m; i++) {
168         if (dj1.in_tree[i] && dj2.in_tree[i]) continue;
169         else {
170             int x = edge[i].first.first, y = edge[i].first.second;
171             LL tmp = 1e18;
172             int l, r;
173             l = min(mp[dj1.f[x]], mp[dj2.f[y]]), r = max(mp[dj1.f[x]], mp[dj2.f[y]]);
174             tmp = dj1.dis[x] + dj2.dis[y] + edge[i].second;
175             if (l >= 1 && r <= cnt)
176                 st.update(1, 1, cnt - 1, l, r - 1, tmp);
177             swap(x, y);
178             l = min(mp[dj1.f[x]], mp[dj2.f[y]]), r = max(mp[dj1.f[x]], mp[dj2.f[y]]);
179             tmp = dj1.dis[x] + dj2.dis[y] + edge[i].second;

```

```

180         if(l >= 1 && r <= cnt)
181             st.update(1, 1, cnt - 1, l, r - 1, tmp);
182     }
183 }
184 int cnt_T = 0;
185 while(T--) {
186     cnt_T++;
187     LL ans = 0;
188     cout<<"change edge[x]'s value to y";
189     scanf("%d%d", &x, &y);
190     int l1 = edge[x].first.first, r1 = edge[x].first.second;
191     if(!dj1.in_line[x]) {
192         ans = dj1.dis[n];
193         ans = min(ans, min(dj1.dis[l1] + dj2.dis[r1] + y, dj1.dis[r1] + dj2.dis[l1] +
194                             y));
195         printf("%lld\n", ans);
196     } else {
197         LL ans = dj1.dis[l1] + dj2.dis[r1] + y;
198         ans = min(ans, dj1.dis[r1] + dj2.dis[l1] + y);
199         int now = min(mp[l1], mp[r1]);
200         printf("%lld\n", min(ans, st.query(1, 1, cnt - 1, now)));
201     }
202 }

```

### 3.4 设计和调试分析及测试结果说明

```

please input the points,edges,query numbers:7 11 3
input the order of the both points and edge value:
1 2 3
1 3 4
2 4 5
3 4 7
2 5 6
3 6 10
4 5 8
4 6 9
5 6 8
5 7 11
6 7 8
change edge[x]'s value to y1 1
18
change edge[x]'s value to y6 2
14
change edge[x]'s value to y10 1
10

```

图三：动态 *dijkstra* 算法测试结果

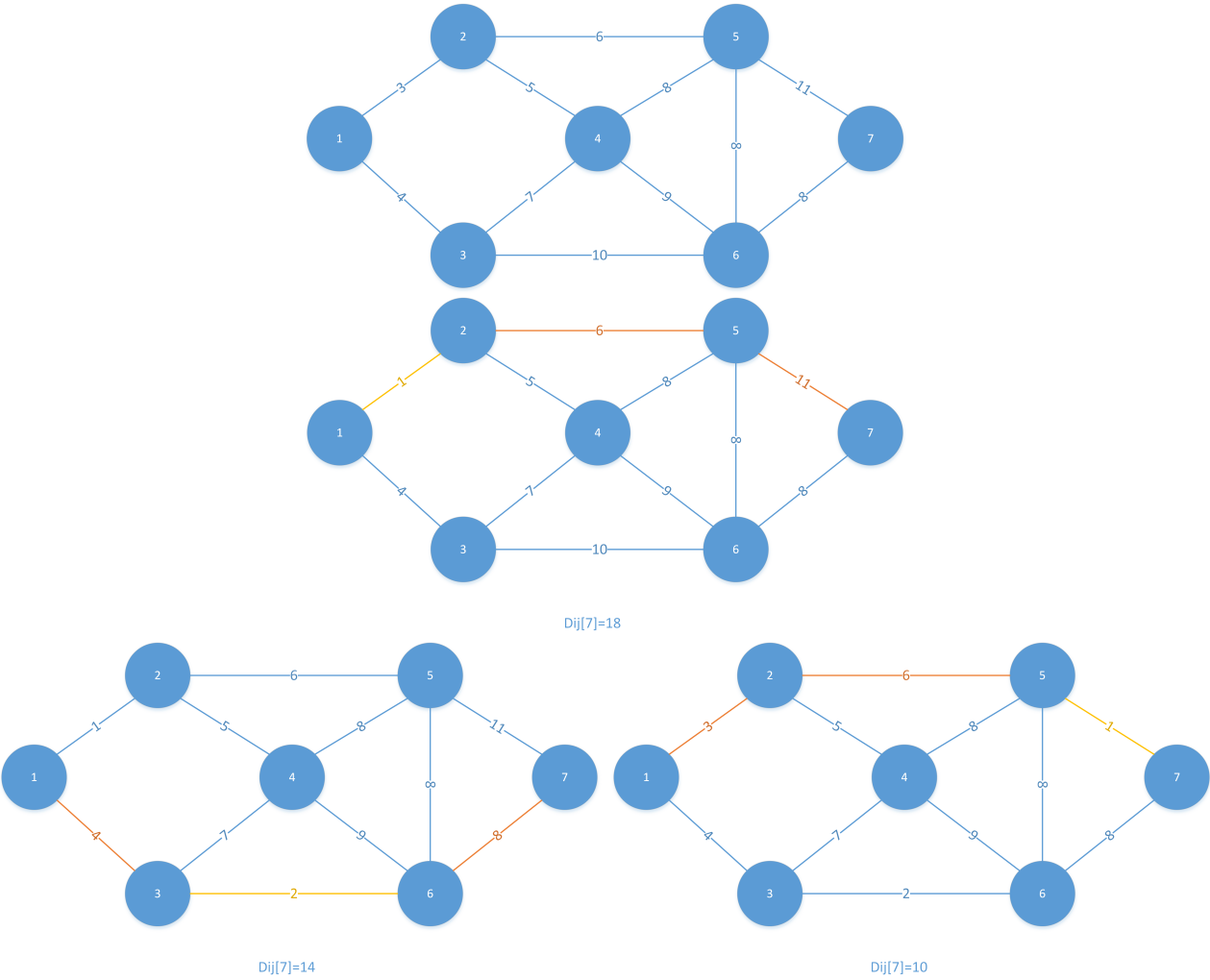
- please input the points edges query numbers 输入图点数，边数和询问次数
- input the order of the both points and edge value 按顺序输入边的关联点和边值
- change edge[x]'s value to y 将边 x 的值改为 y

#### 测试结果说明

输入的图有 7 个点，11 条边，共有 3 次询问



第一次将第一条边的值改为 1  
第二次将第六条边的值改为 2  
第三次将第十条边的值改为 1  
得到的 dij[n] 分别如图所示：



图四：结果图例说明

## 4 Prim 与 Kruskal 最小生成树算法选用

### 4.1 需求分析

众所周知，有两种最小生成树的生成方法分别为 Prim 和 Kruskal, 由于不同的图的边稀疏程度不同，所以采用的两种时间复杂度有所差异，本实验将通过给定的图比较两种算法的优劣，并采用适用的算法，以减少时间浪费的问题。

### 4.2 概述设计

#### 储备知识

首先，最小生成树是一副连通加权无向图中一棵权值最小的生成树。主要可以使用 Prim 和 Kruskal 算法实现，对于稀疏图来说，用 Kruskal 写最小生成树效率更好，加上并查集，可对其进行优化。Kruskal 算法（并查集实现）在使用 Kruskal 实现最小生成树之前，先来看下并查集需要注意的两点：

- 针对树可能会退化为链表的解决方案是，每次合并树时，总是将矮的树挂到高的树下，这种方式称为按秩合并。
- 为了得到的树将更加扁平，加速以后直接或者间接引用节点的速度，Find 时改变每一个节点的引用到根节点，这叫路径压缩。

Kruskal 算法的步骤包括：

- 对所有权值进行从小到大排序（这里对边排序时还需要记录边的索引，这样以边的权值排完序后只改变了权值的索引位置）
- 然后每次选取最小的权值，如果和已有点集构成环则跳过，否则加到该点集中。最终有所有的点集构成的树就是最佳的。

Prim 算法（使用 visited 数组实现）Prim 算法求最小生成树的时候和边数无关，和顶点数有关，所以适合求解稠密网的最小生成树。

Prim 算法的步骤包括：

- 将一个图分为两部分，一部分归为点集 U，一部分归为点集 V，U 的初始集合为 V1，V 的初始集合为 ALL-V1。
- 针对 U 开始找 U 中各节点的所有关联的边的权值最小的那个，然后将关联的节点 Vi 加入到 U 中，并且从 V 中删除（注意不能形成环）。
- 递归执行步骤 2，直到 V 中的集合为空。
- U 中所有节点构成的树就是最小生成树。

#### 算法核心

方法上：Kruskal 在所有边中不断寻找最小的边，Prim 在 U 和 V 两个集合之间寻找权值最小的连接，共同点是构造过程都不能形成环。

时间上：Prim 适合稠密图，复杂度为  $O(n * n)$ ，因此通常使用邻接矩阵储存，复杂度为  $O(e * \log e)$ ，而 Kruskal 多用邻接表，稠密图  $\text{Prim} > \text{Kruskal}$ ，稀疏图  $\text{Kruskal} > \text{Prim}$ 。

空间上：Prim 适合点少边多，Kruskal 适合边多点少。

### 4.3 设计实现

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 const int maxn=50;
4 const int INF=0x3f3f3f3f;
```

```

5     struct node{int from,to, cost;} edge[maxn];
6     int par[maxn];
7     int gr[maxn][maxn];
8     bool visp[maxn];
9     bool acte[maxn][maxn];
10    bool vise[maxn][maxn];
11    int st,tot,n,m;
12    int find(int num)
13    {
14        return par[num]==num?num: par[num]=find(par[num]);
15    }
16    int cmp(const void *a,const void *b)
17    {
18        return (((node*)a)->cost-((node*)b)->cost);
19    }
20
21    void init()
22    {
23        int a,b,c;
24        memset(visp, sizeof(visp), false);
25        memset(acte, sizeof(acte), false);
26        memset(vise, sizeof(vise), false);
27        cin>>n>>m;
28        for (int i=0;i<maxn;i++)
29            for (int j=0;j<maxn;j++)
30                gr[i][j]=INF;
31        cout<<"please input the (v,w) and value:"<<endl;
32        for (int i=1;i<=m;i++)
33        {
34            cin>>a>>b>>c;
35            gr[a][b]=c;
36            gr[b][a]=c;
37        }
38        cout<<"please input the start point:"<<endl;
39        cin>>st;
40        visp[st]=true;
41        for (int i=1;i<=n;i++)
42            if (visp[i]==false&&gr[st][i]!=INF) acte[st][i]=true;
43        tot=1;
44    }
45    void merge()
46    {
47        while (tot<n)
48        {
49            int l=INF,p,v,w;
50            for (int i=1;i<=n;i++)
51                for (int j=1;j<=n;j++)
52                    if (gr[i][j]!=INF&& acte[i][j] && !(visp[i]&&visp[j]))
53                        if (l>gr[i][j])
54                            { if (visp[i]&&!visp[j]) {p=j;l=gr[i][j];v=i;w=j;} if (!visp[i]&&visp[j])
55                                {p=i;l=gr[i][j];v=i;w=j;}}
56            visp[p]=true;

```

```

56         cout<<p;
57         vise[v][w]=true;
58         tot++;
59         for (int i=1;i<=n;i++)
60             if (visp[i]==false&&gr[p][i]!=INF) acte[p][i]=true;
61     }
62 }
63 void prim()
64 {
65     init();
66     int ans=0;
67     merge();
68     for (int i=1;i<=n;i++)
69         for (int j=1;j<=n;j++)
70             if (vise[i][j])
71                 {
72                     cout<<i<<'-'<<j<<endl;
73                     ans+=gr[i][j];
74                 }
75     cout<<"total value:"<<ans;
76 }
77 int main()
78 {
79     cin>>n>>m;
80     if (n*n>m*log(m))
81     {
82         cout<<"Use kruskal algorithm"<<endl;
83         for(int i=0;i<=n;i++)
84             par[i]=i;
85         for(int i=0;i<m;i++)
86             scanf("%d%d%d",&edge[i].from,&edge[i].to,&edge[i].cost);
87         qsort(edge,m,sizeof(node),cmp);
88         int cnt=0,res=0;
89         for(int i=0;i<m;i++)
90         {
91             int sa=find(edge[i].from),sb=find(edge[i].to);
92             if(sa==sb)
93                 continue;
94             res+=edge[i].cost;
95             par[sa]=sb;
96             cnt++;
97             if(cnt==n-1)
98                 break;
99         }
100         if(cnt>=n-1)
101             printf("%d\n",res);
102         else
103             printf("?\n");
104     }
105     else {
106         cout<<"Use prim algorithm"<<endl;
107         prim();

```

```

108     }
109     return 0;
110 }
111 /*
112 input:
113 6 9
114 1 2 6
115 1 3 3
116 2 3 2
117 2 4 5
118 3 4 3
119 4 6 3
120 4 5 2
121 3 5 4
122 5 6 5
123 */

```

#### 4.4 设计调试和测试结果说明

输入:

首先第一行是两个正整数代表图的点数和边数，之后输入每条边的关联点和权值

输出:

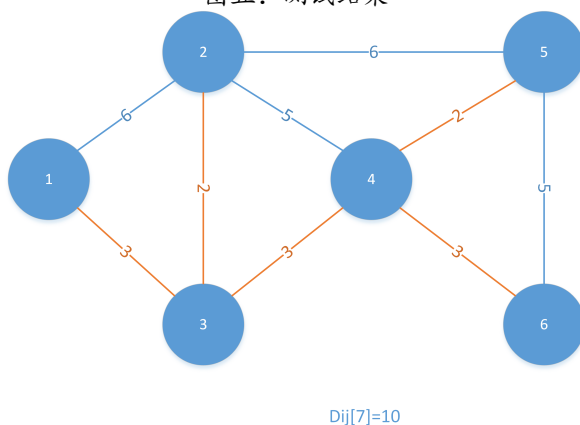
返回使用的方法名称 (Prim or Kruskal)

```

6 9
Use kruskal algorithm
1 2 6
1 3 3
2 3 2
2 4 5
3 4 3
4 6 3
4 5 2
3 5 4
5 6 5
13

```

图五：测试结果



Dij[7]=10

图六：测试结果说明

## 5 二维高斯函数插值

### 5.1 需求分析

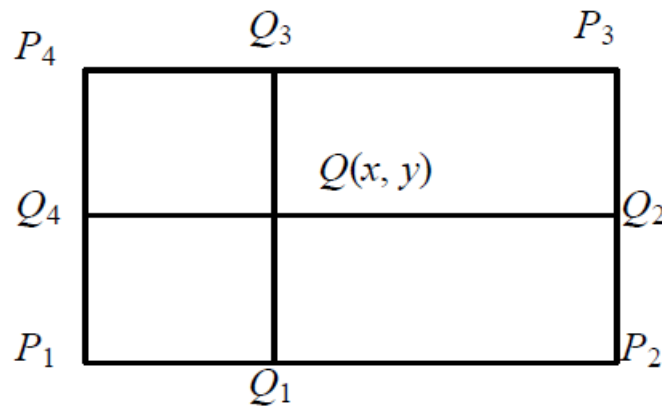
已知二维高斯函数:

$$f(x, y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-\frac{1}{2(1-\rho^2)}\left(\frac{(x-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x-\mu_1)(y-\mu_2)}{\sigma_1\sigma_2} + \frac{(y-\mu_2)^2}{\sigma_2^2}\right)}$$

能否根据一组边界值，直接无误差地得到边界内部所有离散点的值呢？

### 5.2 概述设计

由 Coons 曲面生成原理，记矩形  $\Omega$  的四个顶点  $P_1(x_0, y_0), P_2(x_1, y_0), P_3(x_1, y_1), P_4(x_0, y_1)$ 。设  $Q(x, y)$  是  $\Omega$  内任意一个计算点，经过  $Q$  作矩形平行的平行线，平行线和边界交点分别是  $Q_1(x, y_0), Q_2(x, y_1), Q_3(x, y_1), Q_4(x, y_0)$ ，如果  $F(x, y)$  在边界上连续，则 Coons 插值是： $F(Q) = \sum_{i=1}^4 \frac{A_i + A_{i+1}}{A} f(Q_i) - \sum_{i=1}^4 \frac{A_i}{A} f(P_i)$  其中  $A_1 = A_5 = (x_1 - x)(y_1 - y)$ ,  $A_2 = (x - x_0)(y_1 - y)$ ,  $A_3 = (x - x_0)(y - y_0)$ ,  $A_4 = (x_1 - x)(y - y_0)$ ,  $A_5 = A_1$ ,  $A = (x_1 - x_0)(y_1 - y_0)$ , 从而  $A = A_1 + A_2 + A_3 + A_4$



图六: Coons 曲面生成区域

由于已知边界条件，得到的中心点的坐标对应的  $f(x, y)$ ，可以通过去密集的离散点，得到整个边界内的分布。

### 5.3 设计实现

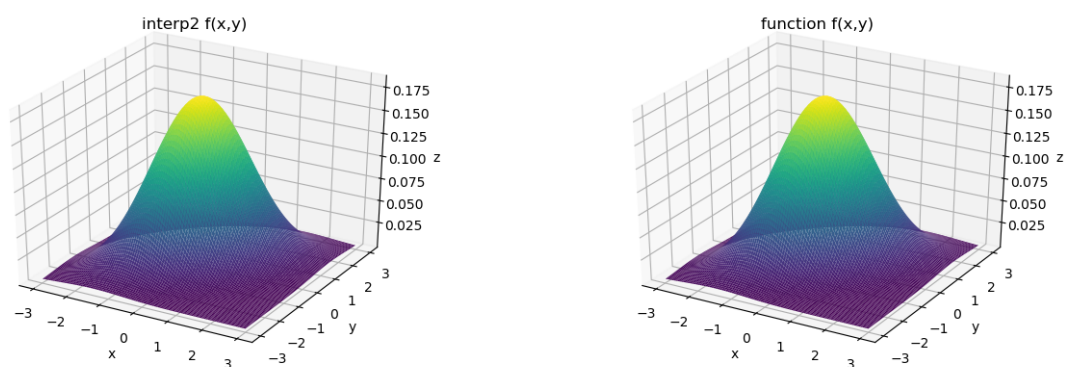
```
1 from mpl_toolkits import mplot3d
2 import matplotlib.pyplot as plt
3 import matplotlib
4 import numpy as np
5 import math
6 sigma1=float(input(' 1: '))
7 mu1=float(input(' 1: '))
8 sigma2=float(input(' 2: '))
9 mu2=float(input(' 2: '))
10 rho=float(input(' : '))
11 x0=float(input('x0: '))
12 y0=float(input('y0: '))
13 x1=float(input('x1: '))
14 y1=float(input('y1: '))
15 def r(x, y):
16     return np.sqrt(x**2+y**2)
17 def p(xi, yi):
```

```

18     return np.log(1/(2*math.pi*sigma1*sigma2*math.sqrt(1-rho**2))*np.exp(-1/(2*math.sqrt(1-rho
    **2))*((xi-mu1)/sigma1)**2+((yi-mu2)\
19     /sigma2)**2-2*rho*(xi-mu1)*(yi-mu2)/(sigma1*sigma2))))
20 def f(x,y):
21     g=0
22     A=(x1-x0)*(y1-y0)
23     Ai=[]
24     fq=[]
25     fp=[]
26     Ai=Ai+[(x1-x)*(y1-y)]+[(x-x0)*(y1-y)]+[(x-x0)*(y-y0)]+[(x1-x)*(y-y0)]
27     fq=fq+[p(x,y0)]+[p(x1,y)]+[p(x,y1)]+[p(x0,y)]
28     fp=fp+[p(x0,y0)]+[p(x1,y0)]+[p(x1,y1)]+[p(x0,y1)]
29     for i in range(4):
30         g=g+(Ai[i]+Ai[(i+1)%4])*fq[i]-Ai[i]*fp[i]
31     return np.exp(g/A)
32 x = np.linspace(x0+0.01,x1-0.01,100)
33 y = np.linspace(x0+0.01,y1-0.01,100)
34 X, Y = np.meshgrid(x, y)
35 Z = f(X,Y)
36 plt.figure(1)
37 ax = plt.axes(projection='3d')
38 ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
39 ax.set_title('interp2 f(x,y)')
40 ax.set_xlabel('x')
41 ax.set_ylabel('y')
42 ax.set_zlabel('z')
43 x = np.linspace(x0+0.01,x1-0.01,100)
44 y = np.linspace(x0+0.01,y1-0.01,100)
45 X, Y = np.meshgrid(x, y)
46 Z = np.exp(p(X,Y))
47 plt.figure(2)
48 ax = plt.axes(projection='3d')
49 ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
50 ax.set_title('function f(x,y)')
51 ax.set_xlabel('x')
52 ax.set_ylabel('y')
53 ax.set_zlabel('z')
54 plt.show()

```

## 5.4 测试结果与说明



图七：插值绘图和函数绘图的比较 根据绘出的  $[-3\ 3\ -3\ 3]$  之间的图可以看出，插值和函数的绘图较为吻合，在一定程度上反映了理论的正确性。

参考文献：二元正态分布函数 (Coons 曲面法) 插值研究邱钧孙洪泉韩伟工程图学学报 2002.Nov.3 2-6



## 6 找对象算法

### 6.1 需求分析

在一定的匹配问题下，如何才能使尽量多的对象能够进行匹配，并且达到最优呢？极大匹配 (Maximal Matching) 是指在当前已完成的匹配下，无法再通过增加未完成匹配的边的方式来增加匹配的边数。最大匹配 (maximum matching) 是所有极大匹配当中边数最大的一个匹配。选择这样的边数最大的子集称为图的最大匹配问题。如果一个匹配中，图中的每个顶点都和图中某条边相关联，则称此匹配为完全匹配，也称作完备匹配。求二分图最大匹配可以用最大流 (Maximal Flow) 或者匈牙利算法 (Hungarian Algorithm)

### 6.2 概述设计

设  $M$  是图  $G=(V,E)$  的一个匹配， $v_i \in V$ 。若  $v_i$  与  $M$  中的边相关联，则称  $v_i$  是  $M$  饱和点，否则称  $v_i$  为  $M$  非饱和点。如果  $G$  中每个顶点都是  $M$  饱和点，则称  $M$  为  $G$  的完美匹配。设  $M$  是  $G$  的一个匹配， $P$  是  $G$  的一条链。如果  $P$  的边交替地一条是  $M$  中的边，一条不是  $M$  中的边，则称  $P$  为  $M$  交错链。类似地，我们可以定义  $G$  的交错圈。易知， $G$  的交错圈一定是偶圈。一条连接两个不同的  $M$  非饱和点的  $M$  交错链称为  $M$  增广链。两个集合  $S_1$  与  $S_2$  的“异或”操作  $S_1 \oplus S_2$  是指集合  $S_1 \oplus S_2 = (S_1 \cup S_2) - (S_1 \cap S_2)$ 。容易看出，设  $M$  是  $G$  的匹配， $P$  是  $G$  中的  $M$  增广链，则  $M \oplus P$  也是  $G$  的匹配，而且可以证明， $G$  中匹配  $M$  是最大匹配当且仅当  $G$  中没有  $M$  增广链。如果一个图是连通的，可以用如下的方法判定是否是二分图：在图中任选一顶点  $v$ ，定义其距离标号为 0，然后把它的邻接点的距离标号均设为 1，接着把所有标号为 1 的邻接点均标号为 2（如果该点未标号的话），如图所示，以此类推。标号过程可以用一次 BFS 实现。标号后，所有标号为奇数的点归为  $X$  部，标号为偶数的点归为  $Y$  部。接下来，二分图的判定就是依次检查每条边，看两个端点是否是一个在  $X$  部，一个在  $Y$  部。如果一个图不连通，则在每个连通块中作判定。

选择边数最大的子图称为图的最大匹配问题 (maximal matching problem) 如果一个匹配中，图中的每个顶点都和图中某条边相关联，则称此匹配为完全匹配，也称作完备匹配。图中所示为一个最大匹配，但不是完全匹配。由增广路的定义可以推出下述三个结论：

- $P$  的路径长度必定为奇数，第一条边和最后一条边都不属于  $M$ ，因为两个端点分属两个集合，且未匹配。
- $P$  经过取反操作可以得到一个更大的匹配  $M'$ 。
- $M$  为  $G$  的最大匹配当且仅当不存在相对于  $M$  的增广路径。

### 6.3 设计实现

```
1 import itertools
2 import numpy as np
3 from numpy import random
4 from scipy.optimize import linear_sum_assignment
5
6 class TaskAssignment:
7
8     # 类初始化，需要输入参数有任务矩阵以及分配方式，其中分配方式有两种，全排列方法或匈牙利方法。
9     all_permutationHungary
10    def __init__(self, task_matrix, mode):
11        self.task_matrix = task_matrix
12        self.mode = mode
13        if mode == 'all_permutation':
14            self.min_cost, self.best_solution = self.all_permutation(task_matrix)
15        if mode == 'Hungary':
16            self.min_cost, self.best_solution = self.Hungary(task_matrix)
```

```
16
17 # 全排列方法
18 def all_permutation(self, task_matrix):
19     number_of_choice = len(task_matrix)
20     solutions = []
21     values = []
22     for each_solution in itertools.permutations(range(number_of_choice)):
23         each_solution = list(each_solution)
24         solution = []
25         value = 0
26         for i in range(len(task_matrix)):
27             value += task_matrix[i][each_solution[i]]
28             solution.append(task_matrix[i][each_solution[i]])
29         values.append(value)
30         solutions.append(solution)
31     min_cost = np.min(values)
32     best_solution = solutions[values.index(min_cost)]
33     return min_cost, best_solution
34
35 # 匈牙利方法
36 def Hungary(self, task_matrix):
37     b = task_matrix.copy()
38     # 行和列减0
39     for i in range(len(b)):
40         row_min = np.min(b[i])
41         for j in range(len(b[i])):
42             b[i][j] -= row_min
43     for i in range(len(b[0])):
44         col_min = np.min(b[:, i])
45         for j in range(len(b)):
46             b[j][i] -= col_min
47     line_count = 0
48     # 线数目小于矩阵长度时, 进行循环
49     while (line_count < len(b)):
50         line_count = 0
51         row_zero_count = []
52         col_zero_count = []
53         for i in range(len(b)):
54             row_zero_count.append(np.sum(b[i] == 0))
55         for i in range(len(b[0])):
56             col_zero_count.append((np.sum(b[:, i] == 0)))
57     # 划线的顺序(分行或列)
58     line_order = []
59     row_or_col = []
60     for i in range(len(b[0]), 0, -1):
61         while (i in row_zero_count):
62             line_order.append(row_zero_count.index(i))
63             row_or_col.append(0)
64             row_zero_count[row_zero_count.index(i)] = 0
65         while (i in col_zero_count):
66             line_order.append(col_zero_count.index(i))
67             row_or_col.append(1)
```

```

68         col_zero_count[col_zero_count.index(i)] = 0
69     # 画线覆盖, 并得到行减最小值, 列加最小值后的矩阵0
70     delete_count_of_row = []
71     delete_count_of_col = []
72     row_and_col = [i for i in range(len(b))]
73     for i in range(len(line_order)):
74         if row_or_col[i] == 0:
75             delete_count_of_row.append(line_order[i])
76         else:
77             delete_count_of_col.append(line_order[i])
78     c = np.delete(b, delete_count_of_row, axis=0)
79     c = np.delete(c, delete_count_of_col, axis=1)
80     line_count = len(delete_count_of_row) + len(delete_count_of_col)
81     # 线数目等于矩阵长度时, 跳出
82     if line_count == len(b):
83         break
84     # 判断是否画线覆盖所有, 若覆盖, 进行加减操作0
85     if 0 not in c:
86         row_sub = list(set(row_and_col) - set(delete_count_of_row))
87         min_value = np.min(c)
88         for i in row_sub:
89             b[i] = b[i] - min_value
90         for i in delete_count_of_col:
91             b[:, i] = b[:, i] + min_value
92         break
93     row_ind, col_ind = linear_sum_assignment(b)
94     min_cost = task_matrix[row_ind, col_ind].sum()
95     best_solution = list(task_matrix[row_ind, col_ind])
96     return min_cost, best_solution
97
98
99 # 生成开销矩阵
100 rd = random.RandomState(10000)
101 task_matrix = rd.randint(0, 100, size=(5, 5))
102 # 用全排列方法实现任务分配
103 ass_by_per = TaskAssignment(task_matrix, 'all_permutation')
104 # 用匈牙利方法实现任务分配
105 ass_by_Hun = TaskAssignment(task_matrix, 'Hungary')
106 print('cost matrix = ', '\n', task_matrix)
107 print('全排列方法任务分配: ')
108 print('min cost = ', ass_by_per.min_cost)
109 print('best solution = ', ass_by_per.best_solution)
110 print('匈牙利方法任务分配: ')
111 print('min cost = ', ass_by_Hun.min_cost)
112 print('best solution = ', ass_by_Hun.best_solution)

```

#### 6.4 分析调试与测试结果

```
cost matrix =  
[[12 53 31 40 47]  
 [ 2 21 72 61 17]  
 [70 72 85 54 39]  
 [93 34 62 75 51]  
 [76 14 15  7 72]]  
全排列方法任务分配:  
min cost = 113  
best solution = [31, 2, 39, 34, 7]  
匈牙利方法任务分配:  
min cost = 113  
best solution = [31, 2, 39, 34, 7]
```

图八：找对象算法结果（全排列和匈牙利算法）

结果说明：

cost matrix: 常量矩阵，随机生成， $a_{ij}$  代表  $i$  与  $j$  的边权值。

根据这个邻接矩阵，用全排列和匈牙利算法求得两种最小的权值 cost。

并在 best solution list 中给出具体的边是哪一条。