计算机网络网络层实验

•

1.实验目的

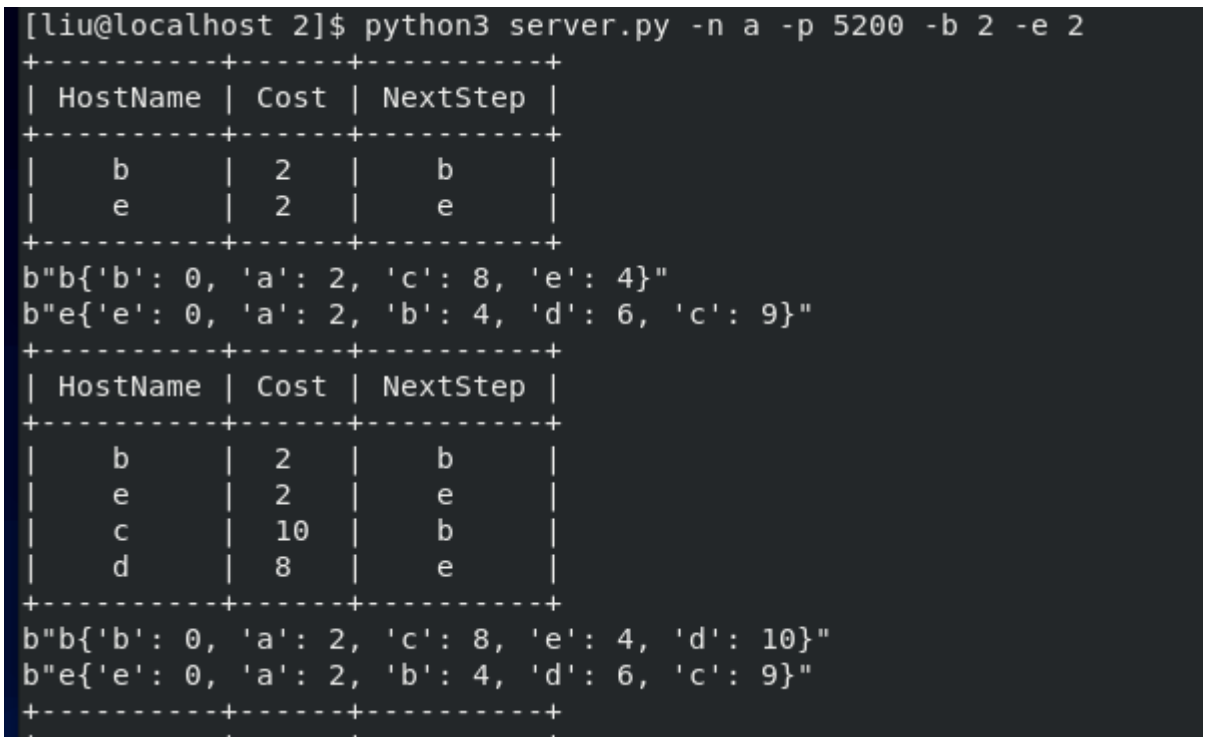
学习和掌握距离向量算法

2.实验内容

2.1任务一

2.1.1各客户机运行及收敛截图如下

客户机1



客户机2

```
[liu@localhost 2]$ python3 server.py -p 5200 -n b -e 6 -c 8 -a 2
b"a{'a': 0, 'b': 2, 'e': 2}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  8   |    c     |
|    e     |  4   |    a     |
+----------+------+----------+
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 12}"
b"e{'e': 0, 'a': 2, 'b': 4, 'd': 6, 'c': 9}"
b"a{'a': 0, 'b': 2, 'e': 2, 'c': 10, 'd': 8}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  8   |    c     |
|    e     |  4   |    a     |
|    d     |  10  |    e     |
+----------+------+----------+
```

客户机3

```
[liu@localhost 2]$ python3 server.py -n c -p 5200 -b 8 -d 3
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  8   |    b     |
|    d     |  3   |    d     |
|    a     |  10  |    b     |
|    e     |  12  |    b     |
+----------+------+----------+
b"d{'d': 0, 'c': 3, 'e': 6, 'b': 11, 'a': 13}"
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4, 'd': 10}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  8   |    b     |
|    d     |  3   |    d     |
|    a     |  10  |    b     |
|    e     |  9   |    d     |
+----------+------+----------+
```

客户机4

```
[liu@localhost 2]$ python3 server.py -n d -p 5200 -c 3 -e 6
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 12}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    e     |  6   |    e     |
|    b     |  11  |    c     |
|    a     |  13  |    c     |
+----------+------+----------+
b"e{'e': 0, 'a': 2, 'b': 4, 'd': 6, 'c': 9}"
b"c{'c': 0, 'b': 8, 'd': 3, 'a': 10, 'e': 9}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    e     |  6   |    e     |
|    b     |  10  |    e     |
|    a     |  8   |    e     |
+----------+------+----------+
```

客户机5

```
[liu@localhost 2]$ python3 server.py -n e -p 5200 -a 2 -b 6 -d 6

b"a{'a': 0, 'b': 2, 'e': 2}"
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4}"
b"d{'d': 0, 'c': 3, 'e': 6, 'b': 11, 'a': 13}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    b     |  4   |    a     |
|    d     |  6   |    d     |
|    c     |  9   |    d     |
+----------+------+----------+
b"a{'a': 0, 'b': 2, 'e': 2, 'c': 10, 'd': 8}"
b"b{'b': 0, 'a': 2, 'c': 8, 'e': 4, 'd': 10}"
b"d{'d': 0, 'c': 3, 'e': 6, 'b': 10, 'a': 8}"
```

2.1.2实验代码（任务一任务二任务三代码相同）

```python
import socket
import os
import sys
import argparse
import threading
import time
import prettytable as pt
#此处用来设置各个客户机的ip地址
#似乎更恰当的方式是定义一个类，这种写法比较丑陋。。
```

```
ipaddrs=
{'a':'192.168.1.201','b':'192.168.1.202','c':'192.168.1.203','d':'192.168.1.204','e':
'192.168.1.205'}
#记录到达每一个客户机的花费
cost={}
#记录临近的客户机
user=[]
#记录每一个客户机的下一条地址
next={}
#记录可以到达的位置
host=[]
#调用argparse来处理命令行参数
def parserinit(parser):
    #指定客户机的名字
    parser.add_argument("-n","--name",help="input the name")
    #指定客户机的端口
    parser.add_argument("-p","--port",type=int,help="bind the port")
    #到达不同客户机的花费
    parser.add_argument("-a","--costa",type=int,help="cost of a")
    parser.add_argument("-b","--costb",type=int,help="cost of a")
    parser.add_argument("-c","--costc",type=int,help="cost of c")
    parser.add_argument("-d","--costd",type=int,help="cost of d")
    parser.add_argument("-e","--coste",type=int,help="cost of e")

def main():
    #处理命令行参数
    parser = argparse.ArgumentParser()
    parserinit(parser)
    args = parser.parse_args()
    init(args)
    #socket绑定与设置
    server = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    server.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    address = (("",args.port))
    server.bind(address)
    #一个线程用来发送消息
    thread1 = sendThread(server,args.name,address)
    #一个线程用来接受消息
    thread2 = recvThread(server,args.name)
    thread1.start()
    thread2.start()

def init(args):
    cost[args.name] = 0
    if(args.costa):
        cost['a'] = args.costa
        user.append('a')
        next['a'] = 'a'
        host.append('a')
    if(args.costb):
        cost['b'] = args.costb
        user.append('b')
        next['b'] = 'b'
```

```
        host.append('b')
    if(args.costc):
        cost['c'] = args.costc
        user.append('c')
        next['c'] = 'c'
        host.append('c')
    if(args.costd):
        cost['d'] = args.costd
        user.append('d')
        next['d'] = 'd'
        host.append('d')
    if(args.coste):
        cost['e'] = args.coste
        user.append('e')
        next['e'] = 'e'
        host.append('e')

class recvThread(threading.Thread):
    #初始化
    def __init__(self,server,name):
        threading.Thread.__init__(self)
        self.server = server
        self.name = name
    def run(self):
        while True:
            #接受消息
            message,addr = self.server.recvfrom(1024)
            #调试用。。。。
            print(message)
            #获得客户机的名字，用于下面的处理
            name = message[0]
            name = chr(name)
            #将字符串转化为字典
            message = eval(message[1:])
            #遍历字典
            for key in message:
                #之前存储过信息
                if key in cost:
                    #到发送者那里去，得到的都是第一手消息，所以只要不是不可达，就更新距离
                    if key == self.name:
                        if message[key] != 9999:
                            cost[name] = message[key]
                            next[name] = name
                    else:
                        #如果距离更短，则更新短的距离
                        if cost[name] + message[key] < cost[key]:
                            cost[key] = cost[name] + message[key]
                            next[key] = name
                        #如果距离不短，但是是来自下一跳的最新的消息，就进行更新
                        elif next[key] == name:
                            cost[key] = cost[name] + message[key]
                            next[key] = name
                else:
```

```
                    cost[key] = cost[name] + message[key]
                    host.append(key)
                    next[key] = name

class sendThread(threading.Thread):
    def __init__(self,server,name,address):
        threading.Thread.__init__(self)
        #设置时间
        self.interval = 20
        self.stopped = False
        self.server = server
        self.name = name
        self.address = address
    def run(self):
        while not self.stopped:
            time.sleep(self.interval)
            #开始发送信息
            sendmsg(self.server,self.name,self.address)
#发送信息
def sendmsg(server,name,address):
    for eachone in user:
        ncost = cost.copy()
        if eachone != name:
            for key in ncost:
                #如果到达临近节点需要中转，那么就发送不可达
                #因为临近节点可以从中转处获得正确的长度
                if key == eachone:
                    if next[key] != eachone:
                        ncost[key] = 9999
    message = name + str(ncost)
    server.sendto(message.encode(),(ipaddrs[eachone],5200))
    tb = pt.PrettyTable()
    tb.field_names = ["HostName","Cost","NextStep"]
    for eachone in host:
        if eachone != name:
            if cost[eachone] >= 9999:
                tb.add_row([eachone,"none",next[eachone]])
            else:
                tb.add_row([eachone,cost[eachone],next[eachone]])
    print(tb)


if __name__ == '__main__':
    main()

```

2.2任务二

2.2.1运行截图

客户机2路径变短

```
[liu@localhost 2]$ python3 server.py -p 5200 -n b -e 2 -c 8 -a 2
b"e{'e': 0, 'a': 2, 'b': 4, 'd': 6, 'c': 9}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  8   |    c     |
|    e     |  2   |    e     |
|    d     |  8   |    e     |
+----------+------+----------+
```

客户机1重新收敛

```
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  2   |    b     |
|    e     |  2   |    e     |
|    d     |  8   |    e     |
|    c     |  10  |    b     |
+----------+------+----------+
```

客户机2重新收敛

```
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  8   |    c     |
|    e     |  2   |    e     |
|    d     |  8   |    e     |
+----------+------+----------+
```

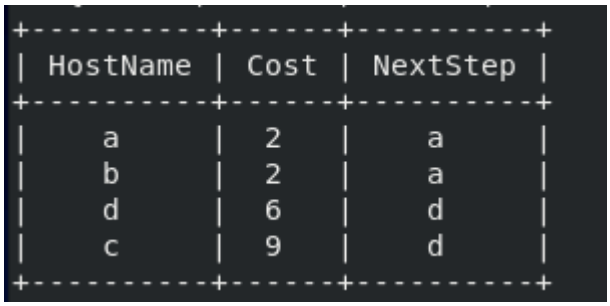
客户机3重新收敛

```
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  8   |    b     |
|    d     |  3   |    d     |
|    e     |  9   |    d     |
|    a     |  10  |    b     |
+----------+------+----------+
```

客户机4收敛

```
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    e     |  6   |    e     |
|    a     |  8   |    e     |
|    b     |  8   |    e     |
+----------+------+----------+
```

客户机5重新收敛

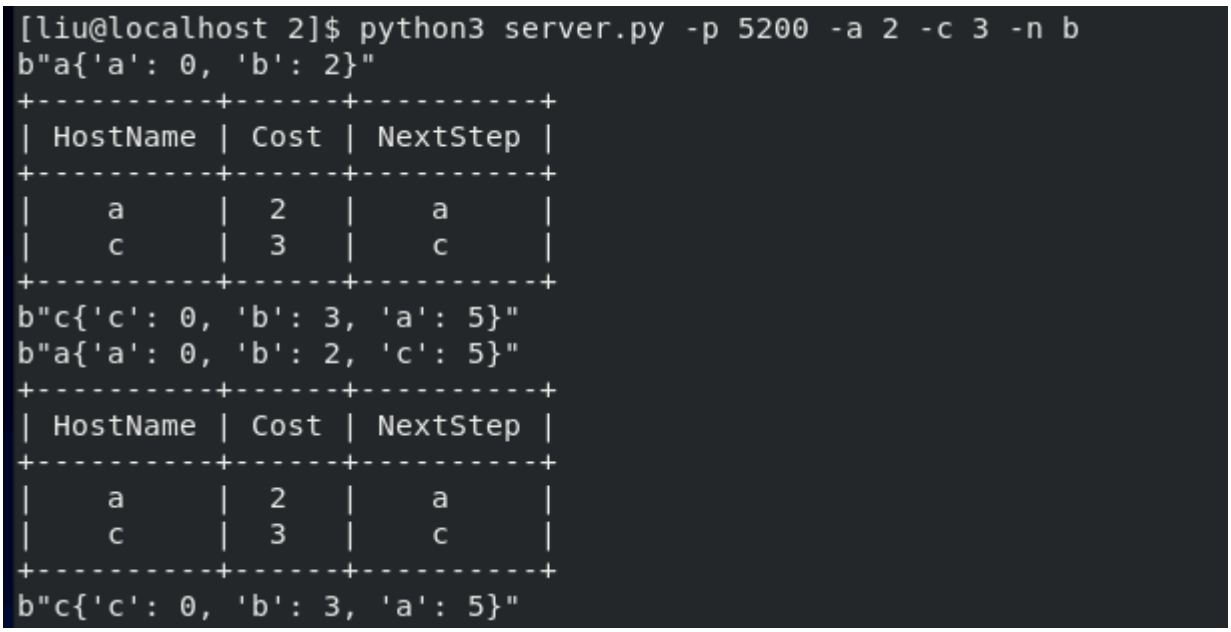


```
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    b     |  2   |    a     |
|    d     |  6   |    d     |
|    c     |  9   |    d     |
+----------+------+----------+
```

2.3任务三

2.3.1运行截图

客户机二完成收敛



```
[liu@localhost 2]$ python3 server.py -p 5200 -a 2 -c 3 -n b
b"a{'a': 0, 'b': 2}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  3   |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 5}"
b"a{'a': 0, 'b': 2, 'c': 5}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  3   |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 5}"
```

终端a与b的连接后

```
[liu@localhost 2]$ python3 server.py -p 5200 -c 3 -n b
b"c{'c': 0, 'b': 3, 'a': 5}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    a     |  8   |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 11}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    a     |  14  |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 17}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    a     |  20  |    c     |
+----------+------+----------+
```

可以看到b到a的距离不断的增大，形成了回环

2.4反向毒化

2.4.1运行截图

建立连接

客户机一

```
[liu@localhost 2]$ python3 server.py -p 5200 -n a -b 2
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  2   |    b     |
+----------+------+----------+
b"b{'b': 0, 'a': 2, 'c': 3}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  2   |    b     |
|    c     |  5   |    b     |
+----------+------+----------+
```

客户机二

```
[liu@localhost 2]$ python3 server.py -p 5200 -c 3 -n b -a 2
b"a{'a': 0, 'b': 2}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  3   |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 9999}"
b"a{'a': 0, 'b': 2, 'c': 9999}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    a     |  2   |    a     |
|    c     |  3   |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 9999}"
b"a{'a': 0, 'b': 2, 'c': 9999}"
```

客户机三

```
[liu@localhost 2]$ python3 server.py -p 5200 -n c -b 3
b"b{'b': 0, 'a': 2, 'c': 3}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  3   |    b     |
|    a     |  5   |    b     |
+----------+------+----------+
b"b{'b': 0, 'a': 2, 'c': 3}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  3   |    b     |
|    a     |  5   |    b     |
+----------+------+----------+
```

断开连接后

客户机二

```
[liu@localhost 2]$ python3 server.py -p 5200 -c 3 -n b
b"c{'c': 0, 'b': 3, 'a': 9999}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    a     | none |    c     |
+----------+------+----------+
b"c{'c': 0, 'b': 3, 'a': 9999}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    c     |  3   |    c     |
|    a     | none |    c     |
+----------+------+----------+
```

客户机三

```
b"b{'b': 0, 'c': 3, 'a': 9999}"
+----------+------+----------+
| HostName | Cost | NextStep |
+----------+------+----------+
|    b     |  3   |    b     |
|    a     | None |    b     |
+----------+------+----------+
```

可以看到此时可以正常显示到达a的信息

2.4.2代码

```python
import socket
import os
import sys
import argparse
import threading
import time
import prettytable as pt
#此处用来设置各个客户机的ip地址
#似乎更恰当的方式是定义一个类，这种写法比较丑陋。。
ipaddrs=
{'a':'192.168.1.201','b':'192.168.1.202','c':'192.168.1.203','d':'192.168.1.204','e':
'192.168.1.205'}
#记录到达每一个客户机的花费
cost={}
#记录临近的客户机
user=[]
#记录每一个客户机的下一条地址
next={}
#记录可以到达的位置
host=[]
#调用argparse来处理命令行参数
def parserinit(parser):
    #指定客户机的名字
```

```
    parser.add_argument("-n","--name",help="input the name")
    #指定客户机的端口
    parser.add_argument("-p","--port",type=int,help="bind the port")
    #到达不同客户机的花费
    parser.add_argument("-a","--costa",type=int,help="cost of a")
    parser.add_argument("-b","--costb",type=int,help="cost of a")
    parser.add_argument("-c","--costc",type=int,help="cost of c")
    parser.add_argument("-d","--costd",type=int,help="cost of d")
    parser.add_argument("-e","--coste",type=int,help="cost of e")

def main():
    #处理命令行参数
    parser = argparse.ArgumentParser()
    parserinit(parser)
    args = parser.parse_args()
    init(args)
    #socket绑定与设置
    server = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    server.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    address = (("",args.port))
    server.bind(address)
    #一个线程用来发送消息
    thread1 = sendThread(server,args.name,address)
    #一个线程用来接受消息
    thread2 = recvThread(server,args.name)
    thread1.start()
    thread2.start()

def init(args):
    cost[args.name] = 0
    if(args.costa):
        cost['a'] = args.costa
        user.append('a')
        next['a'] = 'a'
        host.append('a')
    if(args.costb):
        cost['b'] = args.costb
        user.append('b')
        next['b'] = 'b'
        host.append('b')
    if(args.costc):
        cost['c'] = args.costc
        user.append('c')
        next['c'] = 'c'
        host.append('c')
    if(args.costd):
        cost['d'] = args.costd
        user.append('d')
        next['d'] = 'd'
        host.append('d')
    if(args.coste):
        cost['e'] = args.coste
        user.append('e')
```

```
        next['e'] = 'e'
        host.append('e')

class recvThread(threading.Thread):
    #初始化
    def __init__(self,server,name):
        threading.Thread.__init__(self)
        self.server = server
        self.name = name
    def run(self):
        while True:
            #接受消息
            message,addr = self.server.recvfrom(1024)
            #调试用。。。。
            print(message)
            #获得客户机的名字，用于下面的处理
            name = message[0]
            name = chr(name)
            #将字符串转化为字典
            message = eval(message[1:])
            #遍历字典
            for key in message:
                #之前存储过信息
                if key in cost:
                    #到发送者那里去，得到的都是第一手消息，所以只要不是不可达，就更新距离
                    if key == self.name:
                        if message[key] != 9999:
                            cost[name] = message[key]
                            next[name] = name
                    else:
                        #如果距离更短，则更新短的距离
                        if cost[name] + message[key] < cost[key]:
                            cost[key] = cost[name] + message[key]
                            next[key] = name
                        #如果距离不短，但是是来自下一跳的最新的消息，就进行更新
                        elif next[key] == name:
                            cost[key] = cost[name] + message[key]
                            next[key] = name
                else:
                    cost[key] = cost[name] + message[key]
                    host.append(key)
                    next[key] = name

class sendThread(threading.Thread):
    def __init__(self,server,name,address):
        threading.Thread.__init__(self)
        #设置时间
        self.interval = 20
        self.stopped = False
        self.server = server
        self.name = name
        self.address = address
    def run(self):
```

```
        while not self.stopped:
            time.sleep(self.interval)
            #开始发送信息
            sendmsg(self.server,self.name,self.address)
#发送信息
def sendmsg(server,name,address):
    #对相邻节点发送消息
    for eachone in user:
        ncost = cost.copy()
        #去除自身
        if eachone != name:
            #遍历cost列表
            for key in ncost:
                #去除自身
                if key != name:
                    #不是相邻节点
                    if key != eachone:
                        #由相邻路由器获得的信息即下一跳地址为相邻路由器，就设为不可达
                        if next[key] == eachone:
                            ncost[key] = 9999
                    else:
                #如果到达临近节点需要中转，那么就发送不可达
                #因为临近节点可以从中转处获得正确的长度
                        if next[key] != eachone:
                            ncost[key] = 9999
    #信息，并进行输出
    message = name + str(ncost)
    server.sendto(message.encode(),(ipaddrs[eachone],5200))
    tb = pt.PrettyTable()
    tb.field_names = ["HostName","Cost","NextStep"]
    for eachone in host:
        if eachone != name:
            if cost[eachone] >= 9999:
                tb.add_row([eachone,"none",next[eachone]])
            else:
                tb.add_row([eachone,cost[eachone],next[eachone]])
    print(tb)


if __name__ == '__main__':
    main()

```

2.5思考题

逆向毒化可以避免两个网关之间的路由回路，但在三个网关的情况，还会出现互相欺骗。比如A，B，C两两之间相连，D仅与C相连。当D与C断开连接的时候，A从B知道B可以到D，但B其实是从C就到D，C与D断开连接后，C从A知道A可以到达B，B之后从C知道从C可以到达B。形成了回路。

3.实验心得

在本次实验过程中，通过虚拟机来模拟了路由收敛的过程，以及拓扑变化的过程，制造了路由回路，以及使用逆向毒化算法抑制了路由回路。熟悉了python的多线程，熟悉了服务端建立连接的方式，以及服务端交换信息的方式。熟悉了DV算法以及逆向毒化算法