通信与网络实验三：网络路由层实验

一、网络仿真实现和理解

1、packet.py

函数	功能
init(self, kind, srcAddr, dstAddr,content)	初始化函数，定义数据包的类型、源地址、目的地址和数据包内容
copy(self)	实现数据包的深复制，防止后续链路发送产生混叠问题
isTraceroute(self)	判断数据包是否为Traceroute类型
isRouting(self)	判断数据包是否为路由数据包
getContent(self)	返回数据包内容
addToRoute(self, addr)	更新数据包的传输路径
getRoute(self)	返回数据包的传输路径
animateSend(self, src, dst, latency)	实现仿真界面的动画效果

2、link.py

函数	功能
init(self, e1, e2, l12, l21, latency)	初始化函数，定义链路的两个节点和双向的延时
send_helper(self, packet, src)	实现存在延时的e1->e2或e2->e1的数据包传输
send(self, packet, src)	双线程调用send_helper()实现数据包的发送
recv(self, dst, timeout)	在数据包可以被接收时接收数据包，否则返回空数据包
changeLatency(self, src, c)	更新e1->e2或e2->e1的延时

3、router.py

函数	功能
init(self, addr, heartbeatTime)	初始化函数，定义了路由器的地址、用端口号区分的链路等
changeLink(self, change)	实现链路的增加、移除或者开销的更改。参数change是一个元组，其第一个元素决定执行上述三种功能的哪一个
addLink(self, port, endpointAddr, link, cost)	增加路由器的链路，需要输入新链路的端口号、终点地址和开销。如果对应端口号已经存在链路，则把旧链路移除换成新链路。调用handleNewLink()处理链路新增情况

函数	功能
removeLink(self, port)	移除对应端口号的链路
runRouter(self)	运行路由器的主要函数，监听链路状态改变信息(即链路的增加或移除)并调用 addLink()或 removeLink()，调用 handlePacket()处理路由器收到的数据包
send(self, port, packet)	通过指定端口发送数据包
handlePacket(self, port, packet)	处理路由器收到的数据包
handleNewLink(self, port, endpoint, cost)	处理新增链路，由继承类实现
handleRemoveLink(self, port)	处理被移除的链路，由继承类实现
handleTime(self, timeMillisecs)	根据系统时间周期性发送 traceroute 包，由继承类实现
debugString(self)	生成用于网络仿真调试的字符串

4、client.py

函数	功能
init(self, addr, allClients, sendRate, updateFunction)	初始化函数，定义了客户端地址、发送速率链路改变等基本信息
changeLink(self, change)	客户端增加一条和路由器的链路
handlePacket(self, packet)	处理数据包的函数，忽略路由数据包，收到 traceroute 包则用数据包的路径信息更新
sendTraceroutes(self)	向网络中每个客户端发送 traceroute 包，追踪客户端之间的数据转发路径
handleTime(self, timeMillisecs)	根据系统时间周期性发送 traceroute 包
runClient(self)	运行客户端的主要函数，调用 handlePacket()处理收到的数据包，调用 handlTime()发送 traceroute数据包
lastSend(self)	发送最后一个 traceroute数据包

5、LSP.py

函数	功能
init(self, addr, seqnum, nbcost)	初始化函数，定义了发送链路状态数据包的路由器地址，序列号和相邻节点的链路开销

函数	功能
updateLSP(self, packetIn)	根据接收数据包更新 LSP

二、链路状态法理解与实现

1、初始化函数中变量的作用

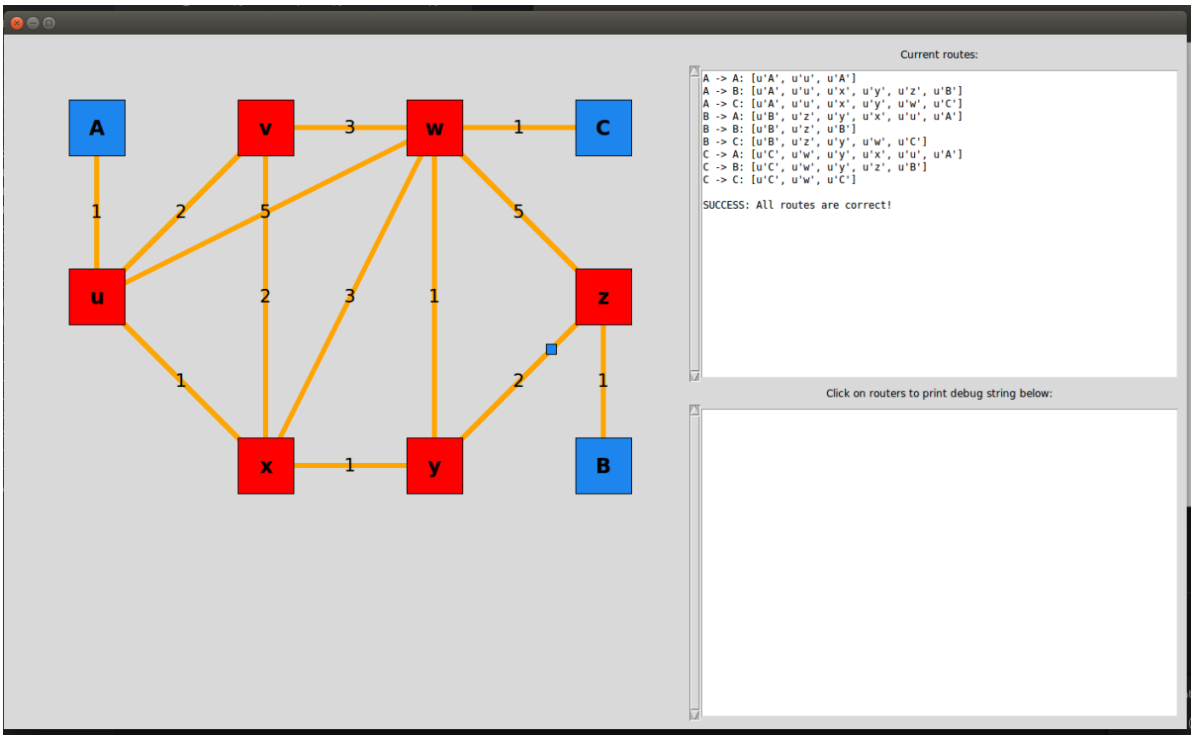
变量	作用
routersLSP	路由器收集的LSP字典，以LSP来源地址作为键值
routersAddr	路由器连接节点的地址，以节点连接的端口号作为键值
routersPort	输入地址，返回路由器和该地址连接的端口号
routersNext	储存当前路由器节点到目的节点下一跳的节点地址，以目的节点地址为键值
routersCost	储存路由器到对应节点的最小开销，以目的节点地址为键值
seqnum	LSP的序列号

2、代码注释及补全

```
def calPath(self):
        # Dijkstra Algorithm for LS routing
        self.setCostMax() # 初始化路由到目的节点的开销，路由器节点自己到自己的最小开销是0，下一跳还是自己
        # put LSP info into a queue for operations
        Q = PriorityQueue() # 初始化优先队列，充当算法中的试探表
        for addr, nbcost in self.routersLSP[self.addr].nbcost.items():
            Q.put((nbcost, addr, addr)) # 把路由器节点（即算法一开始的Next节点）的所有Neighbor压到队列里。下一跳节点和
                                        # 目的节点相同，因为算法刚开始只知道路由器直接相邻的节点信息，只能直接到相邻节点
        while not Q.empty(): # 在试探表不空时重复扩充证实表
            Cost, Addr, Next = Q.get(False) # 取出队列中第一个元素并从队列中删除
            if Addr not in self.routersCost or Cost < self.routersCost[Addr]:
            # 如果目标节点地址不在当前路由器能到达的节点中，或者证实表中到目标节点的开销比Cost的大，则需要更新到该节点的开销和              # 下一跳
                ### TODO: Add two lines code to update Cost and Next for Addr
                self.routersCost[Addr]=Cost # 更新到目的节点的开销
                self.routersNext[Addr]=Next # 更新到目的节点的下一跳
                if Addr in self.routersLSP: # 如果路由器有目的节点的LSP，则把目的节点的所有Neighbor也加入试探表，开销是                                                # 路由器->目的节点+目的节点->Neighbor,下一跳和前往目的节点的下一跳相同
                    for addr_, cost_ in list(self.routersLSP[Addr].nbcost.items()):
                        Q.put((cost_ + Cost, addr_, Next))
```

3、基于正常网络运行链路状态法

运行结果：



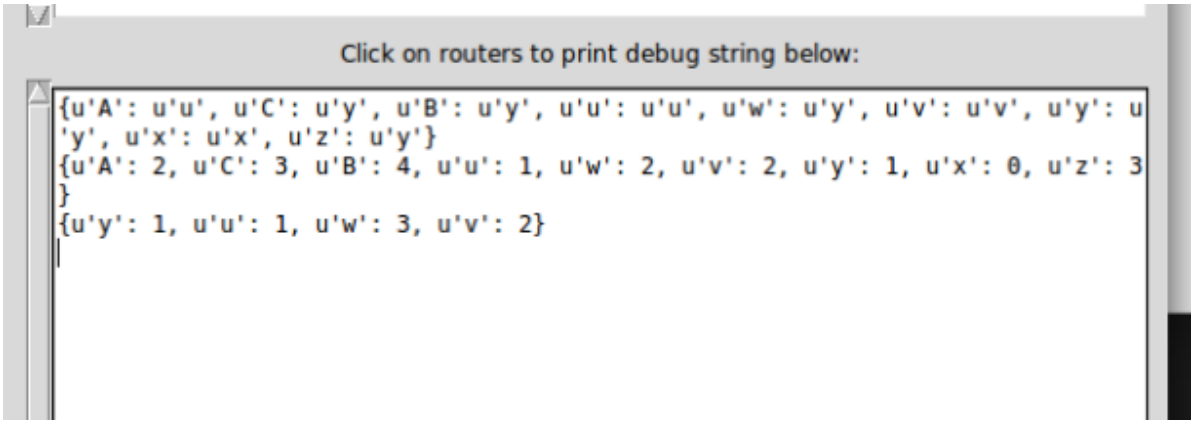
1）客户端之间的最小开销路径：

源客户端-目的客户端	最小开销路径
A → A	A → u → A
A → B	A → u → x → y → z → B
A → C	A → u → x → y → w → C
B → A	B → z → y → x → u → A
B → B	B → z → B
B → C	B → z → y → w → C
C → A	C → w → y → x → u → A
C → B	C → w → y → z → B
C → C	C → w → C

2）路由器 x 的转发表：

目的节点	下一跳转发节点	开销
A	u	2
B	y	4
C	y	3

目的节点	下一跳转发节点	开销
u	u	1
v	v	2
w	y	2
x	x	0
y	y	1
z	y	3



3）路由器 x 的 LSP 信息记录：

相邻节点	开销
y	1
u	1
w	3
v	2

三、链路状态改变实验

1、阅读理解函数 handleRemoveLink()，并对该部分代码注释

```
def handleRemoveLink(self, port):
        """handle removed link"""
        addr = self.routersAddr[port] # 取端口号对应的地址
        self.routersLSP[self.addr].nbcost[addr] = COST_MAX # 链路去掉后，路由器和目标节点不直接相连，开销设为最大值
        self.calPath() # 更新最小开销路径

        content = {} # 初始化空内容
        content["addr"] = self.addr # LSP包发自当前路由器，self.addr即路由器地址
        content["seqnum"] = self.seqnum + 1 # 序列号记录这是路由器发出的第几个包
        content["nbcost"] = self.routersLSP[self.addr].nbcost # 路由器节点的邻居节点开销
        self.seqnum += 1 # 又发出一个包，序列号自加1
        for port1 in self.routersAddr: # 遍历路由器连接的所有节点
```

```
            if port1 != port: # 端口号和移除的链路不同，说明目的节点不是通过删除的链路连接到路由器的
                packet = Packet(Packet.ROUTING, self.addr, self.routersAddr[port1], dumps(content))
                # 打包新的LSP
                self.send(port1, packet) # 发送LSP
        pass
```

2、基于链路故障（移除）网络运行链路状态法

运行结果：

1）客户端之间的最小开销路径：

源客户端-目的客户端	最小开销路径
A → A	A → u → A
A → B	A → u → x → y → z → B
A → C	A → u → x → w → C
B → A	B → z → y → x → u → A
B → B	B → z → B
B → C	B → z → w → C
C → A	C → w → x → u → A
C → B	C → w → z → B
C → C	C → w → C

2）路由器 x 的转发表：

目的节点	下一跳转发节点	开销
A	u	2
B	y	4
C	w	4
u	u	1
v	v	2
w	w	3
x	x	0
y	y	1
z	y	3

3）路由器 x 的 LSP 信息记录：

相邻节点	开销
y	1
u	1
w	3
v	2

3、基于链路新增网络运行链路状态法

运行结果：

1）客户端之间的最小开销路径：

源客户端-目的客户端	最小开销路径
A → A	A → u → A
A → B	A → u → x → z → B
A → C	A → u → x → y → w → C
B → A	B → z → x → u → A
B → B	B → z → B
B → C	B → z → y → w → C
C → A	C → w → y → x → u → A
C → B	C → w → y → z → B
C → C	C → w → C

2）路由器 x 的转发表：

目的节点	下一跳转发节点	开销
A	u	2
B	z	3
C	y	3
u	u	1
v	v	2
w	y	2
x	x	0
y	y	1

目的节点	下一跳转发节点	开销
z	z	2

3）路由器 x 的 LSP 信息记录：

相邻节点	开销
y	1
u	1
w	3
v	2
z	2

4、理解链路状态法处理链路状态改变的过程

链路增加：routersAddr、self.routersPort更新新链路的地址和端口号，routersLSP[self.addr].nbcost[endpoint]=cost即路由器的Neighbor节点增加了一个新节点，并设置开销。之后路由器打包新的LSP，传送给所有路由器连接的节点。

链路减少：用routersAddr取出要移除链路的终点节点地址，设置路由器到这个节点的开销为最大值。调用calPath()更新最短路径。之后路由器打包新的LSP，传送给所有路由器连接的节点（用端口值确保不会把包传给移除链路的节点）。

链路状态更新一般不会把所有最短路径都改变，如上面的最小开销路径和转发表都有不变的项。

四、距离向量法理解与实验

函数	功能
init(self, addr, heartbeatTime)	初始化函数，定义的各变量意义和链路状态法类似
handlePacket(self, port, packet)	处理路由器接收到的数据包。根据数据包的不同类型分别处理。 1.Traceroute包：如果发往数据包的目的节点地址的下一跳节点已知，则向这个下一跳节点连接的端口发送这个Traceroute包 2.路由数据包：更新距离表，如果updateNode()返回None，说明是不相关的数据包，路由器不继续发送。如果不返回None，从返回值得到数据包目的节点地址和从当前路由器到这个目的节点的开销。对所有路由器端口连接的节点发送包含目的节点地址和到这个节点开销的数据包。

函数	功能
updateNode(self, content)	提取数据包内容中的源地址、目的地址和开销。分两种情况讨论。 1.目的地址不在当前路由器储存的已知到达所需开销的节点中并且目的地址不是当前路由器：如果源地址在路由器储存的已知到达所需开销的节点中，更新从路由器到目的节点的开销为 路由器 → 源节点开销+源节点 → 目的节点开销，并返回数据包目的节点地址和从当前路由器到这个目的节点的开销。 2.目的地址在当前路由器储存的已知到达所需开销的节点中：如果源地址也在已知到达所需开销的节点中，并且当前的路由 → 目的节点开销>路由 → 源节点最小开销+源节点 → 目的节点开销或路由器到达目的节点的下一跳是源节点且源节点和目的节点不相同，则更新路由 → 目的节点开销，并更新其对应下一跳为源节点。在此条件下，如果更新后的开销比设置的最大开销还大，则设置为最大开销。
handleNewLink(self, port, endpoint, cost)	根据新增的链路更新距离向量
handleRemoveLink(self, port)	根据移除的链路更新距离向量
handleTime(self, timeMillisecs)	根据系统时间，周期性对路由器连接节点发送路由表
debugString(self)	发送debug的字符串

距离向量法流程图

五、路由选择算法效率比较

使用手机秒表计时

收敛时间（s）	链路正常	链路故障	链路新增
距离向量法	35.08	54.92	55.89
链路状态法	25.23	45.09	45.51

六、实验思考题

(1) 请给出你对 LSP.py 中 LSP 更新函数的执行过程的理解，参见下面代码。

```
def updateLSP(self, packetIn):
        if self.seqnum >= packetIn["seqnum"]: # 收到的LSP序号在路由器储存的LSP之前，说明这个包不带有最新的链路信息，                                    # 不更新LSP并返回false
            return False
        self.seqnum = packetIn["seqnum"]      # 更新序列号
        if self.nbcost == packetIn["nbcost"]: # 邻居节点的开销和之前的LSP相比没变，LSP不进行更新，返回false
            return False
        if self.nbcost != packetIn["nbcost"]: # 邻居节点的开销和之前的LSP相比变化了
            self.nbcost = packetIn["nbcost"]  # 更新路由器储存的LSP中的邻居节点开销
            return True                       # 成功更新LSP，返回true
```

(2) 在距离向量法实现中，下面代码基于 Bellman-Ford 方程进行距离向量更新时， 其中的 self.linksCost[src]能否换成 self.routersCost[src]，请说明原因。

```
if (self.routersCost[dst] > self.linksCost[src] + cost) or
(self.routersNext[dst] == src and src != dst):
```

不可以。self.linksCost[src]描述路由器到源节点的直接链路开销，不一定是self.routersCost[src]描述的路由器到源节点最小链路开销。