目　录

一、实验目的

• 学习和掌握距离向量算法。

二、实验内容

（一）实验要求

1.编程实现并分析以下过程：

（1）模拟路由收敛

（2）模拟拓扑变化

（3）制造路由回路

（4）抑制路由回路

2.程序要求：

（1）使用 python3 编程；

（2）使用 socket 编程实现分布式；

（3）每次迭代后（每隔 Interval，如 30s），各节点输出路由表；

（4）输出收敛后的路由表，即输出每对节点间的最短距离和下一跳。

（二）问题分析

1.简化条件：RIP 协议及距离向量（Distance-Vector）算法过程复杂，细节繁多，这里仅模拟其核心过程，因此有如下简化：

• 对于路由器寻找邻居节点的过程不做模拟，拓扑结构直接通过命令行给出；

• 为方便调试，各个模拟路由器在同一局域网内；

• 当链路发生改变时，在一端输入命令，该端模拟路由器发送消息通知另一端。

2.基本功能：对于每一个模拟路由器，主要有以下三个功能：

（1）监听终端输入，当输入链路变化信息时，更新自己的路由表，并向另一端路由器发送链路变化信息；

（2）定时（RIP 协议规定为 30s）向邻居节点发送路由表（此时模拟路由器为 UDP socket 的

client)；

（3）接收邻居节点发送的信息（此时模拟路由器为 UDP socket 的 server）并进行处理，这里可能接收到两类信息：

①邻居节点发来的路由表，此时需根据收到的路由表更新自己的路由表（DV 算法）；

②邻居节点发来的链路变化信息，此时需根据收到的信息更新路由表和邻居节点信息表。

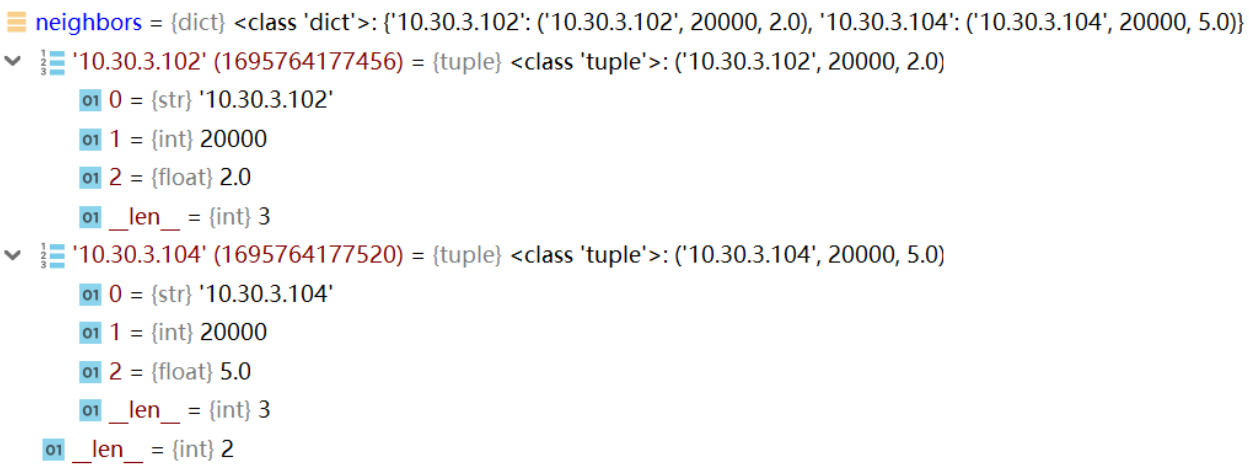
这三个功能同时起作用，因此**用三个线程实现**，其中第二个为定时执行函数的线程。

（三）概要设计

1.数据结构设计

（1）邻居节点信息表（neighbors）：字典，字典的键为邻居节点的 IP 地址，值为邻居节点 IP 地址、端口号、距离组成的元组。

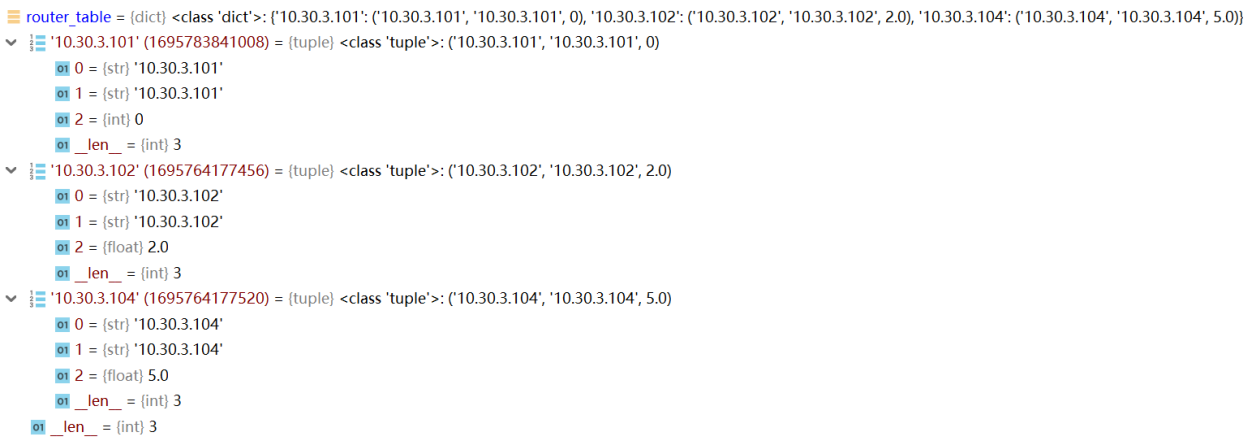
样例如下：



邻居节点信息表信息初始由命令行输入，其作用为初始化路由表、决定发送路由表的对象、更新路由表；当链路发生变化时，邻居节点信息表会发生改变。

（2）路由表（router_table）：字典，字典的键为目的地址，值为目的地址、下一跳、距离组成的元组。

样例如下：



路由表初始只有目的地址为本机和邻居节点的表项，根据邻居节点信息表生成。

2. 通信消息格式设计

（1）路由表通信格式：json 格式，发送端将字典结构的路由表打包，接收端再进行解析获得字典结构。

（2）链路变化信息格式：str（字符串）格式，以[linkchange_type]开头。

链路变化类型	命令行命令	通信消息格式	发送对象
改变边的大小	linkchange <neighbor-ip> <port> <link-cost>	[linkchange]<link-cost>	(<neighbor-ip>,<port>)
取消和这个节点的链接	linkdown <neighbor-ip> <port>	[linkdown]	(<neighbor-ip>,<port>)
增加一条链接	Linkup <neighbor-ip> <port> <link-cost>	[linkup]<link-cost>	(<neighbor-ip>,<port>)

3. 程序基本框架设计

整个工程包含两个文件：router.py、utils.py。其中前者为程序主要功能代码；后者中定义了一系列供 router.py 调用的辅助功能函数。

（1）“router.py”程序可大致分为三个部分（除声明编码方式和引入包依赖）：

- 解析命令行参数并初始化路由表

```
# 从命令行参数中解析监听端口和邻居节点信息表
port, neighbors = parse_argv()

# 根据邻居节点信息表初始化路由表
my_ip = '10.30.3.101' # 本机 IP
router_table = initrt(neighbors, my_ip)

# 输出初始路由表
showrt(router_table, 'Init')
```

- 创建 socket、绑定端口

```
# 创建 socket
router = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# 监听端口
router.bind((my_ip, int(port)))
```

- 定义三个主要功能函数并绑定三个线程

```
# 监听终端输入并处理链路变化信息
def parse_user_input():
    global router_table, neighbors, router
    while True:
        cmd = input()
        if cmd:
            ...


# 定时向邻居节点发送路由表
def broadcast_costs(router, poinsoned_flag=True):
    global router_table, neighbors
    if poinsoned_flag:  # 若使用逆向毒化技术
        ...
    else:  # 若不使用逆向毒化技术
        ...
```

```
# 接收并处理邻居节点发来的信息
def update_costs(router):
    global router_table, neighbors
    while True:
        _data, addr = router.recvfrom(1024)  # 接收消息
        data = _data.decode('utf-8')
        if data:  # 收到信息不为空
            ...


# 绑定三个线程
Thread(target=parse_user_input).start()
RepeatTimer(interval=30, target=broadcast_costs, socket=router).start()
Thread(target=update_costs, args=(router,)).start()
```

（2）“utils.py”程序中定义了以下功能函数：

函数名	函数功能	输入参数	返回值
parse_argv	解析命令行参数	无	监听端口，邻居节点信息表
initrt	初始化路由表	邻居节点信息表，本机 IP 地址	初始化后的路由表
showrt	在终端显示路由表	待显示的路由表，附加信息（路由表状态，如初始、更新、收敛等）	无
data_analysis	解析收到的消息（对于链路更新信息）	收到的消息（已解码）	消息类型、消息内容
linkchange	对链路距离改变做出反应	路由表、邻居节点信息表、改变的链路信息	更新后的路由表、邻居节点信息表
linkdown	对链路断开做出反应		
linkup	对新增链路做出反应		

此外，还包含继承 Thread 类创建的定时执行函数的线程类。

（四）详细设计

1.监听终端输入并处理链路变化信息

•“router.py”中：

```
# 监听终端输入并处理链路变化信息
def parse_user_input():
    global router_table, neighbors, router
    while True:
        cmd = input()
        if cmd:
            argv = cmd.split(' ')  # 按照空格分割，得到参数列表

            # 根据 argv[0]判断链路改变类型，触发不同事件并按格式生成通知信息
            if argv[0] == 'linkchange':
                router_table, neighbors = linkchange(
                        router_table, neighbors, argv[1], int(argv[2]), float(argv[3]))
                send_msg = '[linkchange]' + argv[3]
            elif argv[0] == 'linkdown':
                router_table, neighbors = linkdown(
                                router_table, neighbors, argv[1], int(argv[2]))
                send_msg = '[linkdown]'
            elif argv[0] == 'linkup':
                router_table, neighbors = linkup(
                        router_table, neighbors, argv[1], int(argv[2]), float(argv[3]))
                send_msg = '[linkup]' + argv[3]
            else:  # 若为无效消息，则发送空消息，这样接收端会自动过滤掉
                send_msg = ''

            addr = (argv[1], int(argv[2]))  # 发送地址
            router.sendto(send_msg.encode(), addr) # 发送链路改变消息
            print ('Send link msg to ', addr)
            print('send_msg:' + send_msg)
```

•“utils.py”中：

```
# 对链路距离改变做出反应
def linkchange(router_table, neighbors, host, port, cost):
    neighbors[host] = (host, port, cost)  # 更新邻居信息节点表中对应项的距离
    router_table[host] = (host, host, cost)  # 更新路由表中对应项
    showrt(router_table, 'Change')  # 显示改变后的路由表
    return router_table, neighbors
```

```
# 对链路断开做出反应
def linkdown(router_table, neighbors, host, port):
    router_table[host] = (host, host, float("inf"))  # 更新路由表中对应项距离为无穷大
    neighbors.pop(host)  # 删除邻居信息节点表中对应项
    showrt(router_table, 'Change')  # 显示改变后的路由表
    return router_table, neighbors


# 对新增链路做出反应
def linkup(router_table, neighbors, host, port, cost):
    neighbors[host] = (host, port, cost)  # 在邻居信息节点表新增一项
    router_table[host] = (host, host, cost)  # 更新路由表中对应项
    showrt(router_table, 'Change')  # 显示改变后的路由表
    return router_table, neighbors
```

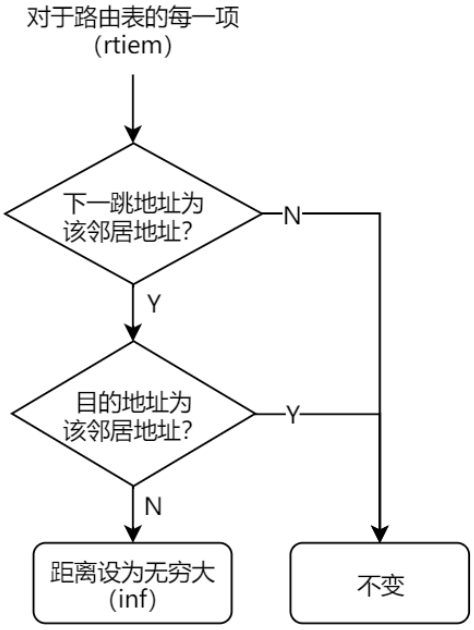
2.定时向邻居节点发送路由表

•“router.py”中：

```
# 定时向邻居节点发送路由表
def broadcast_costs(router, poinsoned_flag=False):
    global router_table, neighbors
    if poinsoned_flag:  # 若使用逆向毒化技术
        # 对于每一个邻居，发送毒化后的路由表
        for neighbor in neighbors.keys():
            _router_table = copy.deepcopy(router_table)  # 拷贝路由表

            # 毒化路由表
            for rtiem in _router_table.keys():
                if _router_table[rtiem][1] == neighbor and rtiem != neighbor:
                    _router_table[rtiem] = (rtiem, neighbor, float("inf"))

            # 发送毒化后的路由表
            addr = (neighbors[neighbor][0], neighbors[neighbor][1])  # 发送地址
            send_json = json.dumps(_router_table).encode('utf-8')  # 将路由表打包成
json 格式
            router.sendto(send_json, addr)  # 发送信息
            print ('Send router_table to ', addr)
            showrt(_router_table, 'Send')  # 显示发送的路由表
```

```
    else:  # 若不使用逆向毒化技术
        for neighbor in neighbors.keys():
            # 对于每一个邻居，直接发送自己的路由表
            addr = (neighbors[neighbor][0], neighbors[neighbor][1])  # 发送地址
            send_json = json.dumps(router_table).encode('utf-8')  # 将路由表打包成
    json 格式
            router.sendto(send_json, addr)  # 发送信息
            print ('Send router_table to ', addr)
            showrt(router_table, 'Send')  # 显示发送的路由表
```

其中路由表毒化算法流程图如下



•“utils.py”中，实现定时执行函数的线程类：

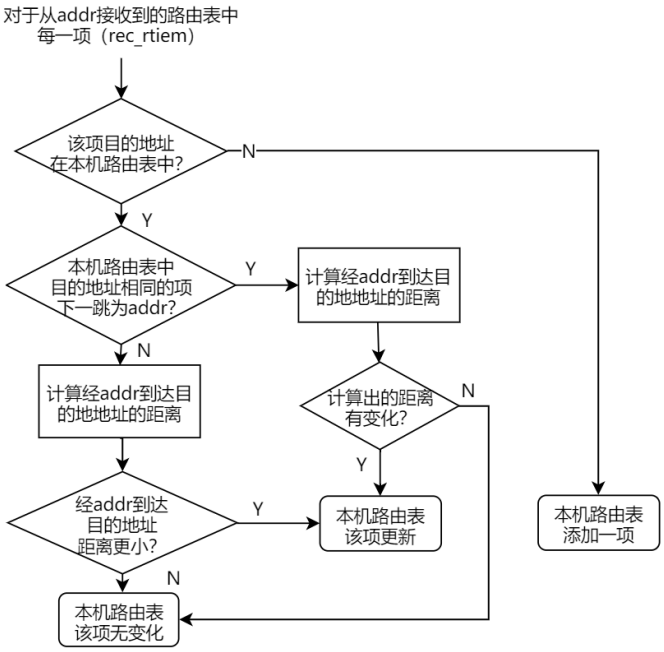
```
class RepeatTimer(Thread):
    def __init__(self, interval, target, socket):
        Thread.__init__(self)
        self.interval = interval  # 发送间隔
        self.target = target
        self.socket = socket
        self.daemon = True
        self.stopped = False

    def run(self):
        while not self.stopped:
            time.sleep(self.interval)
            self.target(self.socket)
```

3.接收并处理邻居节点发来的信息

•“router.py”中：

```
# 接收并处理邻居节点发来的信息
def update_costs(router):
    global router_table, neighbors
    while True:
        _data, addr = router.recvfrom(1024)  # 接收消息
        data = _data.decode('utf-8')
        if data:  # 收到的消息不为空
            if data[0] == '[':  # 收到链路改变信息
                print ('Recv link msg from ', addr)
                msg_type, msg = data_analysis(data)  # 解析信息
                # 按照不同的类型，进行不同的处理
                if msg_type == 'linkchange':
                    router_table, neighbors = linkchange(
                        router_table, neighbors, addr[0], addr[1], float(msg))
                if msg_type == 'linkdown':
                    router_table, neighbors = linkdown(
                        router_table, neighbors, addr[0], addr[1])
                if msg_type == 'linkup':
                    router_table, neighbors = linkup(
                        router_table, neighbors, addr[0], addr[1], float(msg))
            else:  # 收到路由信息，则执行路由表更新算法
                ...（下接）
```

其中路由表更新算法流程图如下：



代码如下：

```
else:  # 收到路由信息
    rec_rt = json.loads(data)  # 将收到路由的信息转为字典格式
    print ('Recv router_table from ', addr)
    showrt(rec_rt, 'Recv')  # 显示收到的路由表
    converge_flag = True  # 是否收敛标志
    for rec_rtiem in rec_rt.keys():  # 对于收到的路由表中的每一项
        if rec_rtiem not in router_table.keys():  # 出现新的目的地址，则加入路由表
            router_table[rec_rtiem] = (
                rec_rtiem,
                addr[0],
                rec_rt[rec_rtiem][2] + neighbors[addr[0]][2])
            converge_flag = False  # 路由表有变化则置收敛标志为 False
        else:  # 已经存在的目的地址
            if router_table[rec_rtiem][1] == addr[0]:  # 若下一跳相同，有变化则更新路
由表
                if rec_rt[rec_rtiem][2] + neighbors[addr[0]][2] \
                        != router_table[rec_rtiem][2]:
                    router_table[rec_rtiem] = (
                        rec_rtiem,
                        addr[0],
                        rec_rt[rec_rtiem][2] + neighbors[addr[0]][2])
                    converge_flag = False  # 路由表有变化则置收敛标志为 False
            else:  # 若下一跳不同，则比较距离决定是否更新
                if rec_rt[rec_rtiem][2] + neighbors[addr[0]][2] \
                        < router_table[rec_rtiem][2]:
                    router_table[rec_rtiem] = (
                        rec_rtiem,
                        addr[0],
                        rec_rt[rec_rtiem][2] + neighbors[addr[0]][2])
                    converge_flag = False  # 路由表有变化则置收敛标志为 False
    showrt(router_table, 'Converge') if converge_flag else showrt(router_table,
'Update')
```

•“utils.py”中：

```
# 对消息进行分析，返回消息类型和消息有效部分
def data_analysis(data):
    _msg_type = data.split(']')  # 按照右中括号切分
    msg_type = _msg_type[0][1:]  # 左侧为消息类型（去除左中括号）
    msg = _msg_type[1]  # 左侧为消息内容
    return msg_type, msg
```

4. 其他功能实现

• 解析命令行参数：

```
# 解析命令行参数，获取监听端口和邻居节点信息表
def parse_argv():
    s = sys.argv[1:]  # 命令行参数列表
    port = int(s.pop(0))  # 首个为监听端口（需转换为 int 型）
    neighbors = {}
    slen = int(len(s) / 3)
    for i in range(0, slen):  # 每相邻三个组成居节点信息表中一项
        neighbors[s[3 * i]] = (s[3 * i], int(s[3 * i + 1]), float(s[3 * i + 2]))
    return port, neighbors
```

• 初始化路由表：

```
# 根据邻居节点初始化路由表
def initrt(neighbors, my_ip):
    rt = {}
    rt[my_ip] = (my_ip, my_ip, 0)  # 到达本机 IP 直接交付，距离为 0
    for neighbor in neighbors.keys():  # 到达邻居节点
        rt[neighbor] = (neighbor, neighbor, neighbors[neighbor][2])
    return rt
```

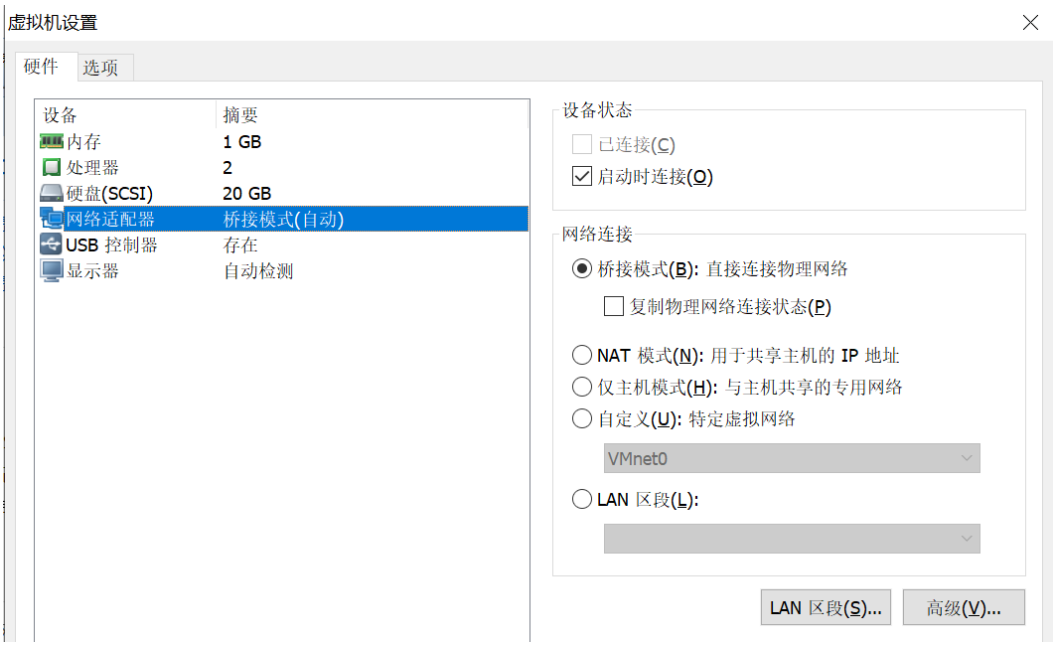
• 显示路由表：

```
# 显示路由表
def showrt(rt, msg):
    tb = pt.PrettyTable()  # 创建 PrettyTable 类
    tb.field_names = ["Destination", "Next Hop", "Cost"]  # 创建表头
    _rt = sorted(rt)  # 按照目的地址从小到大排序
    for ritem in _rt:  # 路由表中的每一项
        tb.add_row(rt[ritem])

    # 根据路由表不同状态，显示不同颜色的路由表
    color = {'Update': '31', 'Converge': '32', 'Change': '33',
             'Init': '34', 'Recv': '35', 'Send': '36'}
    print('\033[1;%s;40m' % color[msg])
    print("[%s] Router Table at " % msg,
          time.strftime('%Y.%m.%d %H:%M:%S ', time.localtime(time.time())))
    print(tb)
    print('\033[0m')
```

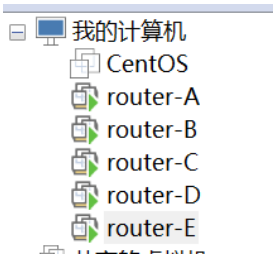
三、实验结果及分析

1. 实验环境搭建

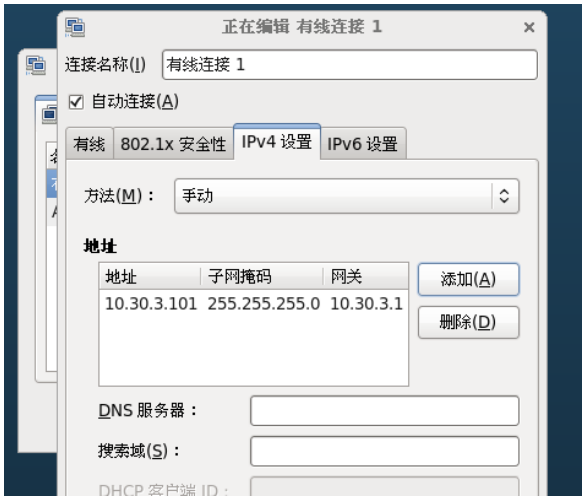
- 将虚拟机网络适配器设置为“桥接模式”；



- 克隆五台虚拟机（链接克隆），分别命名为“router-*”（A-E）；



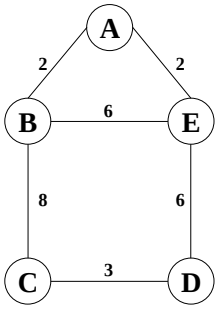
- 更改每台虚拟机的网络设置，将五台虚拟机 IP 分别设置为“10.30.3.10*”（1-5）；



• 用 ping 命令测试虚拟机之间的连通性。

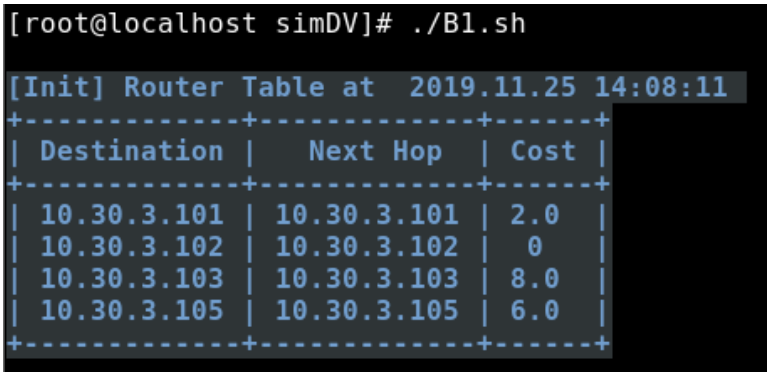
2. 任务 1：模拟路由收敛

已知网络拓扑如下图所示，请使用 DV 算法模拟该网络的迭代收敛过程。

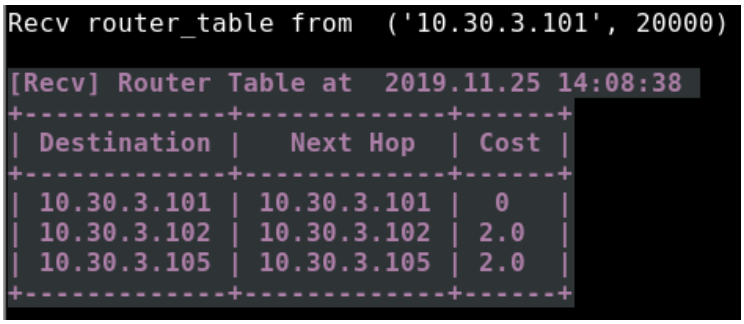


（1）以路由表 B(10.30.3.102)为例进行分析

• 初始 B 的路由表如下：



• B 收到来自 A 的信息：



A 告诉 B “A 跳 E 到达 E 距离为 2”，则 B 可知若先到 A 再到 E 距离为 4（B 到 A 距离为 2），比原先路由表中直接跳 E 的距离小，因此更新路由表：

```
[Update] Router Table at  2019.11.25 14:08:38
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.105 | 10.30.3.101 | 4.0  |
+-------------+-------------+------+
```

- B 收到来自 C 的信息：

```
Recv router_table from  ('10.30.3.103', 20000)

[Recv] Router Table at  2019.11.25 14:08:42
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 10.0 |
| 10.30.3.102 | 10.30.3.102 | 8.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
| 10.30.3.104 | 10.30.3.104 | 3.0  |
| 10.30.3.105 | 10.30.3.102 | 12.0 |
+-------------+-------------+------+
```

C 告诉 B “C 跳 D 到达 D 距离为 3”，而 B 原先路由表无到达 D 的路由，因此新增一项“B 跳 C 到达 D 距离为 11”（B 跳 C 到 C 距离为 8）：

```
[Update] Router Table at  2019.11.25 14:08:42
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.104 | 10.30.3.103 | 11.0 |
| 10.30.3.105 | 10.30.3.101 | 4.0  |
+-------------+-------------+------+
```

- B 收到来自 E 的信息：

```
Recv router_table from  ('10.30.3.105', 20000)

[Recv] Router Table at  2019.11.25 14:08:48
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.101 | 4.0  |
| 10.30.3.103 | 10.30.3.104 | 9.0  |
| 10.30.3.104 | 10.30.3.104 | 6.0  |
| 10.30.3.105 | 10.30.3.105 |  0   |
+-------------+-------------+------+
```

E 告诉 B “E 跳 D 到 D 距离为 6”，而 B 已知 “B 到 E 距离为 6”，则 B 可知 “B 跳 E 到 D 距离为 12”比原先路由距离大，因此不更新路由表：

• B 收到来自 A 的消息：

```
Recv router_table from  ('10.30.3.101', 20000)

[Recv] Router Table at  2019.11.25 14:09:08
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.102 | 10.0 |
| 10.30.3.104 | 10.30.3.105 | 8.0  |
| 10.30.3.105 | 10.30.3.105 | 2.0  |
+-------------+-------------+------+
```

A 告诉 B “A 跳 E 到 D 距离为 8”，而 B 已知 “B 到 A 距离为 2”，则 B 可知 “B 跳 A 到 D 距离为 10”比原先路由距离小，因此更新路由表：

```
[Update] Router Table at  2019.11.25 14:09:08
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.104 | 10.30.3.101 | 10.0 |
| 10.30.3.105 | 10.30.3.101 | 4.0  |
+-------------+-------------+------+
```

• 经几轮迭代后，B 最终达到收敛：

```
[Converge] Router Table at  2019.11.25 14:09:12
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.104 | 10.30.3.101 | 10.0 |
| 10.30.3.105 | 10.30.3.101 | 4.0  |
+-------------+-------------+------+
```

（2）其他各路由器收敛后的路由表结果

①路由器 A(10.30.3.101)：

```
[Converge] Router Table at  2019.11.25 14:09:48
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.102 | 10.0 |
| 10.30.3.104 | 10.30.3.105 | 8.0  |
| 10.30.3.105 | 10.30.3.105 | 2.0  |
+-------------+-------------+------+
```

①路由器 C(10.30.3.103)：

```
[Converge] Router Table at  2019.11.25 14:09:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 10.0 |
| 10.30.3.102 | 10.30.3.102 | 8.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
| 10.30.3.104 | 10.30.3.104 | 3.0  |
| 10.30.3.105 | 10.30.3.104 | 9.0  |
+-------------+-------------+------+
```

①路由器 D(10.30.3.104)：

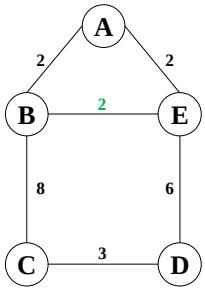
```
[Converge] Router Table at  2019.11.25 14:09:48
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.105 | 8.0  |
| 10.30.3.102 | 10.30.3.105 | 10.0 |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
| 10.30.3.104 | 10.30.3.104 |  0   |
| 10.30.3.105 | 10.30.3.105 | 6.0  |
+-------------+-------------+------+
```

①路由器 E(10.30.3.105)：

```
[Converge] Router Table at  2019.11.25 14:09:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.101 | 4.0  |
| 10.30.3.103 | 10.30.3.104 | 9.0  |
| 10.30.3.104 | 10.30.3.104 | 6.0  |
| 10.30.3.105 | 10.30.3.105 |  0   |
+-------------+-------------+------+
```

3. 任务 2：模拟拓扑变化

在任务 1 的网络收敛后，将 B 和 E 之间的距离更改 6->2（好消息!），模拟该变化导致的重新收敛过程。



• 在路由器 B 执行链路距离改变命令“linkchange 10.30.3.105 20000 2”，路由器 E 同步收到消息：

```
linkchange 10.30.3.105 20000 2

[Change] Router Table at  2019.11.25 14:30:03
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.104 | 10.30.3.101 | 10.0 |
| 10.30.3.105 | 10.30.3.105 | 2.0  |
+-------------+-------------+------+

Send link msg to  ('10.30.3.105', 20000)
send_msg:[linkchange]2
```

```
Recv link msg from  ('10.30.3.102', 20000)

[Change] Router Table at  2019.11.25 14:30:03
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.104 | 9.0  |
| 10.30.3.104 | 10.30.3.104 | 6.0  |
| 10.30.3.105 | 10.30.3.105 |  0   |
+-------------+-------------+------+
```

• D 收到来自 E 的消息：

```
Recv router_table from  ('10.30.3.105', 20000)

[Recv] Router Table at  2019.11.25 14:30:32
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.104 | 9.0  |
| 10.30.3.104 | 10.30.3.104 | 6.0  |
| 10.30.3.105 | 10.30.3.105 |  0   |
+-------------+-------------+------+
```

E 告诉 D “E 跳 B 到 B 距离为 2”，而 D 已知“D 到 E 距离为 6”，则 D 可知“D 跳 E 到 B 距离为 8”比原先路由距离小，因此更新路由表：

```
[Update] Router Table at  2019.11.25 14:30:32
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.105 | 8.0  |
| 10.30.3.102 | 10.30.3.105 | 8.0  |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
| 10.30.3.104 | 10.30.3.104 |  0   |
| 10.30.3.105 | 10.30.3.105 | 6.0  |
+-------------+-------------+------+
```

• B 收到来自 E 的消息：

```
Recv router_table from  ('10.30.3.105', 20000)

[Recv] Router Table at  2019.11.25 14:30:32
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.104 | 9.0  |
| 10.30.3.104 | 10.30.3.104 | 6.0  |
| 10.30.3.105 | 10.30.3.105 |  0   |
+-------------+-------------+------+
```

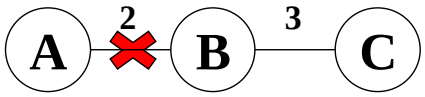
E 告诉 B“E 跳 D 到 D 距离为 6”，而 B 已知“B 到 E 距离为 2”，则 B 可知“B 跳 E 到 D 距离为 8”比原先路由距离小，因此更新路由表：

```
[Update] Router Table at  2019.11.25 14:30:32
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 8.0  |
| 10.30.3.104 | 10.30.3.105 | 8.0  |
| 10.30.3.105 | 10.30.3.105 | 2.0  |
+-------------+-------------+------+
```

A、C 并未受到路由变化的影响，其他路由器仅需一到两次更新路由就重新收敛，可见“好消息传播得快”。

4. 任务 3：制造路由回路

将下图拓扑的 A 和 B 连接断开（坏消息!），模拟该变化导致的重新收敛过程。



• 首先，路由达到收敛：

①路由器 A(10.30.3.101)：

```
[Converge] Router Table at  2019.11.23 20:46:10
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.102 | 5.0  |
+-------------+-------------+------+
```

②路由器 B(10.30.3.102)：

```
[Converge] Router Table at  2019.11.23 20:46:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

③路由器 C(10.30.3.103)：

```
[Converge] Router Table at  2019.11.23 20:47:10
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 5.0  |
| 10.30.3.102 | 10.30.3.102 | 3.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
+-------------+-------------+------+
```

• 然后，在路由器 A 执行链路断开命令“linkdown 10.30.3.102 20000”，路由器 B 同步收到消息：

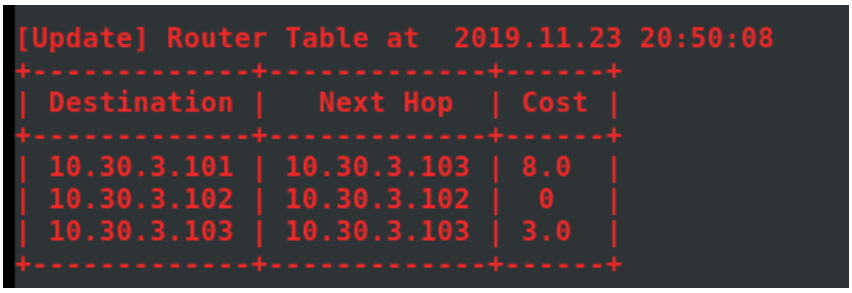
```
linkdown 10.30.3.102 20000

[Change] Router Table at  2019.11.23 20:49:55
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | inf  |
| 10.30.3.103 | 10.30.3.102 | 5.0  |
+-------------+-------------+------+

Send link msg to  ('10.30.3.102', 20000)
send_msg:[linkdown]
```

```
Recv link msg from  ('10.30.3.101', 20000)

[Change] Router Table at  2019.11.23 20:49:55
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | inf  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

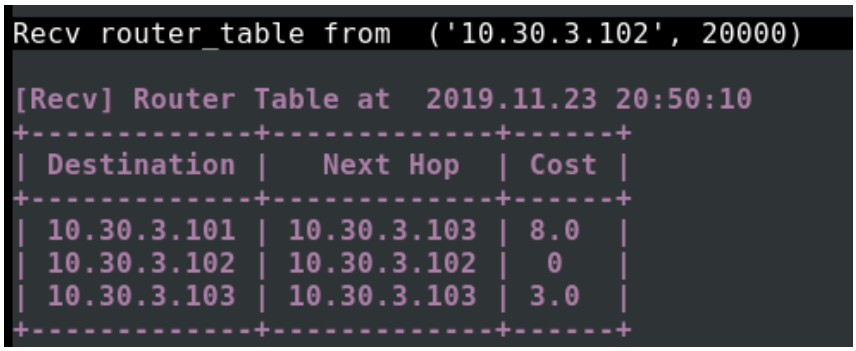
• B 收到来自 C 的消息

```
Recv router_table from  ('10.30.3.103', 20000)

[Recv] Router Table at  2019.11.23 20:50:08
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 5.0  |
| 10.30.3.102 | 10.30.3.102 | 3.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
+-------------+-------------+------+
```

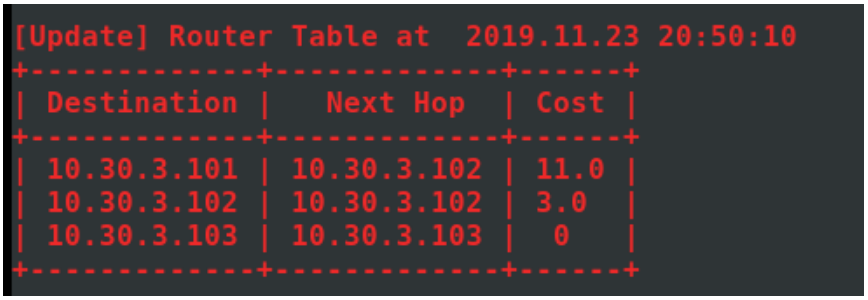
C 告诉 B “C 跳 B 到 A 距离为 5”，而 B 已知 “B 到 C 距离为 3”，则 B 可知 “B 跳 C 到 A 距离为 8”（但实际上已经不可达了）比原先路由距离（inf）小，因此更新路由表：



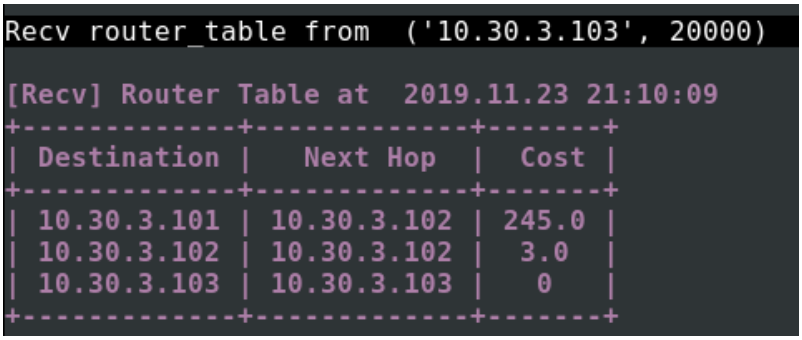
• C 收到来自 B 的消息：



B 告诉 C “B 跳 C 到 A 距离为 8”，而 C 已知 “C 跳 B 到 A 距离为 8”，则 C 可知 “C 跳 B 到 A 距离为 11”（但实际上已经不可达了），由于下一跳相同，不需要比较距离直接更新路由表：



• 之后，B 收到来自 C 的消息、C 收到来自 B 的消息……两个过程交替进行，到 A 的距离逐渐增大（即无穷计算现象）。一段时间后，仍在计数更新（“坏消息传播得慢”）：



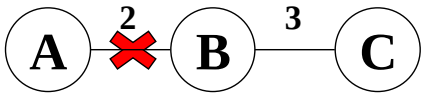
```
[Update] Router Table at  2019.11.23 21:10:09
+-------------+-------------+-------+
| Destination |   Next Hop  |  Cost |
+-------------+-------------+-------+
| 10.30.3.101 | 10.30.3.103 | 248.0 |
| 10.30.3.102 | 10.30.3.102 |   0   |
| 10.30.3.103 | 10.30.3.103 |  3.0  |
+-------------+-------------+-------+
```

```
Recv router_table from  ('10.30.3.102', 20000)

[Recv] Router Table at  2019.11.23 21:10:41
+-------------+-------------+-------+
| Destination |   Next Hop  |  Cost |
+-------------+-------------+-------+
| 10.30.3.101 | 10.30.3.103 | 254.0 |
| 10.30.3.102 | 10.30.3.102 |   0   |
| 10.30.3.103 | 10.30.3.103 |  3.0  |
+-------------+-------------+-------+
```

```
[Update] Router Table at  2019.11.23 21:10:41
+-------------+-------------+-------+
| Destination |   Next Hop  |  Cost |
+-------------+-------------+-------+
| 10.30.3.101 | 10.30.3.102 | 257.0 |
| 10.30.3.102 | 10.30.3.102 |  3.0  |
| 10.30.3.103 | 10.30.3.103 |   0   |
+-------------+-------------+-------+
```

5. 任务 4：解决路由回路（逆向毒化技术）

如果使用逆向毒化技术，重新模拟 A 和 B 链接断开所导致的重新收敛过程。



• 首先，路由达到收敛：

①路由器 A(10.30.3.101)：

```
[Converge] Router Table at  2019.11.23 21:42:46
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.102 | 5.0  |
+-------------+-------------+------+
```

②路由器 B(10.30.3.102)：

```
[Converge] Router Table at  2019.11.23 21:42:17
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 2.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

③路由器 C(10.30.3.103)：

```
[Converge] Router Table at  2019.11.23 21:43:16
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 5.0  |
| 10.30.3.102 | 10.30.3.102 | 3.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
+-------------+-------------+------+
```

• 然后，在路由器 A 执行链路断开命令“linkdown 10.30.3.102 20000”，路由器 B 同步收到消息：

```
linkdown 10.30.3.102 20000

[Change] Router Table at  2019.11.23 21:44:26
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | inf  |
| 10.30.3.103 | 10.30.3.102 | 5.0  |
+-------------+-------------+------+

Send link msg to  ('10.30.3.102', 20000)
send_msg:[linkdown]
```

```
Recv link msg from  ('10.30.3.101', 20000)

[Change] Router Table at  2019.11.23 21:44:26
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | inf  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

• B 收到来自 C 的消息：

此时收到的路由表为 C 毒化后的路由表。由于 C 中路由表项“C 跳 B 到达 A 距离为 5”该项路由信息来自 B，因此发送给 B 时会将距离设为不可达（inf）。

```
Recv router_table from  ('10.30.3.103', 20000)

[Recv] Router Table at  2019.11.23 21:44:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | inf  |
| 10.30.3.102 | 10.30.3.102 | 3.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
+-------------+-------------+------+
```

```
[Converge] Router Table at  2019.11.23 21:44:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | inf  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

• C 收到来自 B 的消息：

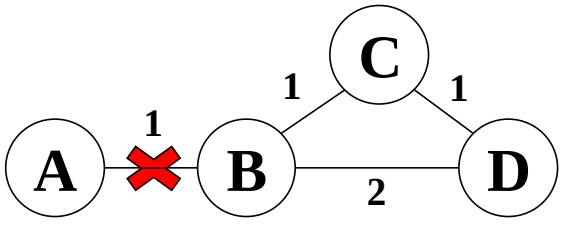
```
Recv router_table from  ('10.30.3.102', 20000)

[Recv] Router Table at  2019.11.23 21:44:46
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | inf  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 3.0  |
+-------------+-------------+------+
```

```
[Update] Router Table at  2019.11.23 21:44:46
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | inf  |
| 10.30.3.102 | 10.30.3.102 | 3.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
+-------------+-------------+------+
```

可见，路由回路没有出现，路由表很快收敛。

6. 举例说明为什么逆向毒化不能杜绝回路生成

找到反例网络，其拓扑如下：



• 首先，路由达到收敛：

①路由器 A(10.30.3.101)：

```
[Converge] Router Table at  2019.11.24 15:08:20
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | 1.0  |
| 10.30.3.103 | 10.30.3.102 | 2.0  |
| 10.30.3.104 | 10.30.3.102 | 3.0  |
+-------------+-------------+------+
```

②路由器 B(10.30.3.102)：

```
[Converge] Router Table at  2019.11.24 15:09:12
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | 1.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 | 2.0  |
+-------------+-------------+------+
```

③路由器 C(10.30.3.103)：

```
[Converge] Router Table at  2019.11.24 15:07:44
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 2.0  |
| 10.30.3.102 | 10.30.3.102 | 1.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
| 10.30.3.104 | 10.30.3.104 | 1.0  |
+-------------+-------------+------+
```

④路由器 D(10.30.3.104)：

```
[Converge] Router Table at  2019.11.24 15:07:50
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.103 | 3.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 |  0   |
+-------------+-------------+------+
```

• 然后，在路由器 B 执行链路断开命令“linkdown 10.30.3.101 20000”，路由器 A 同步收到消息：

```
linkdown 10.30.3.101 20000

[Change] Router Table at  2019.11.24 15:17:59
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 | inf  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 | 2.0  |
+-------------+-------------+------+

Send link msg to  ('10.30.3.101', 20000)
send_msg:[linkdown]
```

```
Recv link msg from  ('10.30.3.102', 20000)

[Change] Router Table at  2019.11.24 15:17:59
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.101 |  0   |
| 10.30.3.102 | 10.30.3.102 | inf  |
| 10.30.3.103 | 10.30.3.102 | 2.0  |
| 10.30.3.104 | 10.30.3.102 | 3.0  |
+-------------+-------------+------+
```

• B 收到来自 D 的消息：

由于 D 到达 A 的下一跳为 C（非 D），因此该项并不会被毒化。

```
Recv router_table from  ('10.30.3.104', 20000)

[Recv] Router Table at  2019.11.24 15:18:20
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.103 | 3.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 |  0   |
+-------------+-------------+------+
```

D 告诉 B “D 跳 C 到 A 距离为 3”，而 B 已知 “B 到 D 距离为 2”，则 C 可知 “C 跳 D 到 A 距离为 5”（但实际上已经不可达了），比原先路由距离（inf）小，因此更新路表：

```
[Update] Router Table at  2019.11.24 15:18:20
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.104 | 5.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 | 2.0  |
+-------------+-------------+------+
```

• C 收到来自 B 的消息：

由于 B 到达 A 的下一跳为 D（非 C），因此该项并不会被毒化。

```
Recv router_table from  ('10.30.3.102', 20000)

[Recv] Router Table at  2019.11.24 15:18:27
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.104 | 5.0  |
| 10.30.3.102 | 10.30.3.102 |  0   |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 | 2.0  |
+-------------+-------------+------+
```

B 告诉 C“B 跳 D 到 A 距离为 5”，而 C 已知“C 到 B 距离为 1”，则 C 可知“C 跳 B 到 A 距离为 6”（但实际上已经不可达了），由于下一跳相同，不需要比较距离直接更新路由表：

```
[Update] Router Table at  2019.11.24 15:18:27
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 6.0  |
| 10.30.3.102 | 10.30.3.102 | 1.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
| 10.30.3.104 | 10.30.3.104 | 1.0  |
+-------------+-------------+------+
```

- D 收到来自 C 的消息：

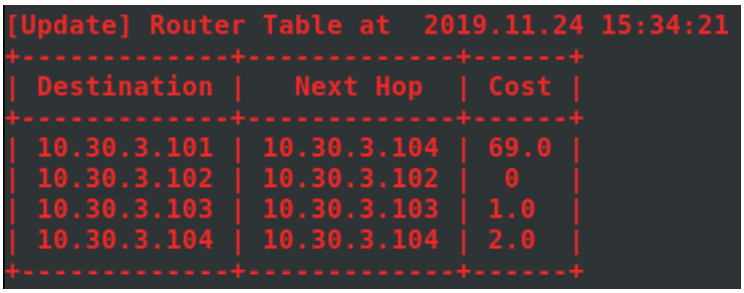
由于 C 到达 A 的下一跳为 B（非 D），因此该项并不会被毒化。

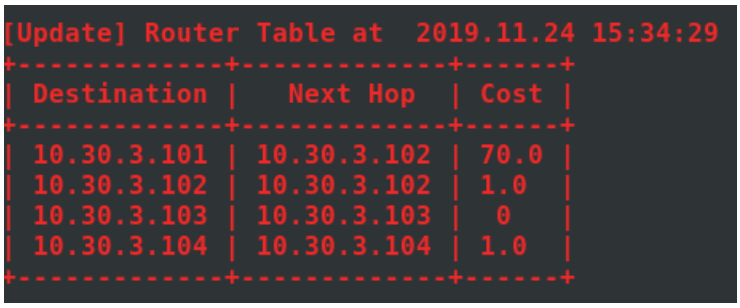
```
Recv router_table from  ('10.30.3.103', 20000)

[Recv] Router Table at  2019.11.24 15:18:52
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.102 | 6.0  |
| 10.30.3.102 | 10.30.3.102 | 1.0  |
| 10.30.3.103 | 10.30.3.103 |  0   |
| 10.30.3.104 | 10.30.3.104 | 1.0  |
+-------------+-------------+------+
```

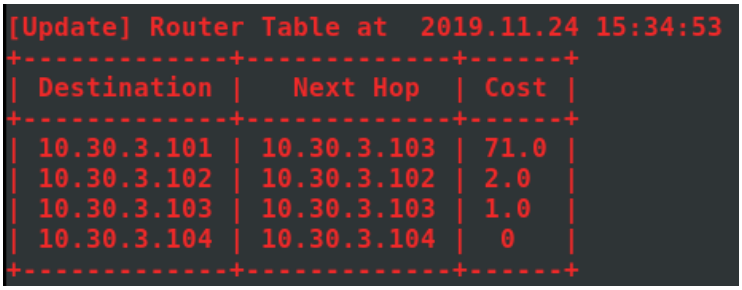
C 告诉 D“C 跳 B 到 A 距离为 6”，而 D 已知“D 到 C 距离为 1”，则 D 可知“D 跳 C 到 A 距离为 7”（但实际上已经不可达了），由于下一跳相同，不需要比较距离直接更新路由表：

```
[Update] Router Table at  2019.11.24 15:18:52
+-------------+-------------+------+
| Destination |   Next Hop  | Cost |
+-------------+-------------+------+
| 10.30.3.101 | 10.30.3.103 | 7.0  |
| 10.30.3.102 | 10.30.3.102 | 2.0  |
| 10.30.3.103 | 10.30.3.103 | 1.0  |
| 10.30.3.104 | 10.30.3.104 |  0   |
+-------------+-------------+------+
```

- 之后，B 收到来自 D 的消息、C 收到来自 B 的消息、D 收到来自 C 的消息……三个过程交替进行，到 A 的距离逐渐增大（即无穷计算现象）。一段时间后，仍在计数更新（逆向毒化未能解决）：


[Update] Router Table at 2019.11.24 15:34:21
Destination	Next Hop	Cost
10.30.3.101	10.30.3.104	69.0
10.30.3.102	10.30.3.102	0
10.30.3.103	10.30.3.103	1.0
10.30.3.104	10.30.3.104	2.0


[Update] Router Table at 2019.11.24 15:34:29
Destination	Next Hop	Cost
10.30.3.101	10.30.3.102	70.0
10.30.3.102	10.30.3.102	1.0
10.30.3.103	10.30.3.103	0
10.30.3.104	10.30.3.104	1.0


[Update] Router Table at 2019.11.24 15:34:53
Destination	Next Hop	Cost
10.30.3.101	10.30.3.103	71.0
10.30.3.102	10.30.3.102	2.0
10.30.3.103	10.30.3.103	1.0
10.30.3.104	10.30.3.104	0