Circular DHT Report

1. **Environment**：Win10，python 3.7
2. **Tool**：PyCharm
3. **Design Architecture**：

   1） 同时运行三个线程，分别用来运行 tcp、udp 和 ping 操作。

```
# run three threads
tcpThread = threading.Thread(target=set_tcp_port, args=(peer + basePort,))
udpThread = threading.Thread(target=set_udp_port, args=(peer + basePort,))
pingThread = threading.Thread(target=ping, args=(peer,))
tcpThread.start()
udpThread.start()
pingThread.start()
```

```
def ping(peer):
    global successor_first, successor_second, pingSeq_first, pingSeq_second
    request_seq_first = 0
    request_seq_second = 0
    while running:
        # send to successor first
        message = Message().send_ping_message("pingRequest", peer, request_seq_first, 0)
        address = (host, successor_first + basePort)
        send_udp_message(message, address)
        pingSeq_first.append(request_seq_first)
        request_seq_first += 1
        # send to successor second
        message = Message().send_ping_message("pingRequest", peer, request_seq_second, 1)
        address = (host, successor_second + basePort)
        send_udp_message(message, address)
        pingSeq_second.append(request_seq_second)
        request_seq_second += 1

        time.sleep(5)
        pingCheck(peer)
```

   2） udp 协议用于传输 ping 消息，使结点与其邻居保持联系，包括 ping request 和 ping response；tcp 协议用于传输其他消息，例如：插入结点、移出结点、查找结点等。

```
def set_udp_port(port):
    udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    address = (host, port)
    udp.bind(address)
    while running:
        data, _ = udp.recvfrom(1024)
        receive_udp_message(data, port - basePort)
    udp.close()


def set_tcp_port(port):
    tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    address = (host, port)
    tcp.bind(address)
    tcp.listen(5)
    while running:
        client, _ = tcp.accept()
        data = client.recv(1024)
        receive_tcp_message(data, port - basePort)
        client.close()
    tcp.close()
```

3) 设置一个 tcp 客户机和一个 udp 客户机来发出消息，一个 tcp 服务器和一个 udp 服务器来处理消息。

发送消息：

```
def send_udp_message(data, address):
    udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp.sendto(data, address)
    udp.close()


def send_tcp_message(data, address):
    tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp.connect(address)
    tcp.send(data)
    tcp.close()
```

接收消息：

```
def receive_udp_message(data, peer):
    global pingSeq_first, pingSeq_second, predecessor_first, predecessor_second
    m = Message()
    m.receive_message(data)
```

```
def receive_tcp_message(data, peer):
    global successor_first, successor_second, pingSeq_first, pingSeq_second, running
    m = Message()
    m.receive_message(data)
```

4) 发出的消息封装在一个 Message 类中，属性包括消息类型、消息结点和消息参数，便于其他结点收到消息后对消息进行处理。

```
t message format
class Message:
    def __init__(self):
        self.type = ""
        self.peer = -1
        self.data = ""
        self.seqNumber = -1
        self.successor = -1

    def send_ping_message(self, type, peer, seqNumber, successor):
        message = "{0}-{1}-{2}-{3}".format(type, peer, seqNumber, successor)
        return message.encode()

    def send_other_message(self, type, peer, data):
        message = "{0}-{1}-{2}".format(type, peer, data)
        return message.encode()

    def receive_message(self, message):
        m = message.decode()
        str = m.split("-")
        if len(str) == 3:
            self.type = str[0]
            self.peer = int(str[1])
            self.data = str[2]
        elif len(str) == 4:
            self.type = str[0]
            self.peer = int(str[1])
            self.seqNumber = int(str[2])
            self.successor = int(str[3])
```

5) 每个结点维护他的两个前驱和后继的信息，便于信息的传递。

```
successor_first, successor_second = -1, -1  # the first and second successor
predecessor_first, predecessor_second = -1, -1  # the first and second predecessor
```

6) 每个结点维护其第一后继和第二后继的 seq number 列表，发出 ping request 则将对应 seq 存入列表，收到 ping response 则将对应 seq 从列表中删除，并删除小于该 seq number 的所有值。如果列表中保存的 seq number 数量大于 2，说明该列表对应的后继已经不再存在，则结点更新其后继。

```
# receive ping response message
elif m.type == "pingResponse":
    print("A ping response was received from peer {}".format(m.peer))
    if successor_first == m.peer and m.seqNumber in pingSeq_first:
        pingSeq_first.remove(m.seqNumber)
        pingSeq_first = [i for i in pingSeq_first if i > m.seqNumber]  # delete the early seqNumber
    elif successor_second == m.peer and m.seqNumber in pingSeq_second:
        pingSeq_second.remove(m.seqNumber)
        pingSeq_second = [i for i in pingSeq_second if i > m.seqNumber]
```

```
# check whether the peer is alive, if unreceived ping number more than 3, then the peer is no longer alive
def pingCheck(peer):
    global successor_first, successor_second, pingSeq_first, pingSeq_second
    if len(pingSeq_first) > 2:
        print("peer {} is not alive.".format(successor_first))
        message = Message().send_other_message("quit", peer, 0)
        address = (host, basePort + successor_second)
        send_tcp_message(message, address)
    elif len(pingSeq_second) > 2:
        print("peer {} is not alive.".format(successor_second))
        message = Message().send_other_message("quit", peer, 1)
        address = (host, basePort + successor_first)
        send_tcp_message(message, address)
```

4. Install manual

点击 start.bat 即可启动程序，初始化 16 个结点。

5. User manual

1) Initialize 文件夹中保存要初始化的结点信息。

1) 初始化完成后，ping 操作每隔 5s 自动进行。

```
Initialised Peer 30. Successor first is 31, successor second is 36
Please input 'insert peer i' or 'remove peer i':
A ping request was received from peer 20.
A ping request was received from peer 25.
A ping response was received from peer 31
A ping response was received from peer 36
A ping request was received from peer 20.
A ping request was received from peer 25.
A ping response was received from peer 31
A ping response was received from peer 36
A ping request was received from peer 20.
A ping request was received from peer 25.
A ping response was received from peer 31
A ping response was received from peer 36
A ping request was received from peer 20.
```

2) 可输入“insert peer x”来插入结点，程序会自动找到合适的位置，初始化新的结点并通知相关结点更改后继，提醒插入成功。如果插入的是已存在的结点，那么插入失败。

a) 插入成功，系统自动弹出新增的结点窗口，相关的结点会更换后继

```
Peer 80 has successfully insert!
My first successor is now 80.
My second successor is now 1.
```

b) 插入失败，提醒 peer x 已存在：

```
insert peer 48A ping request was received from peer 30.

Peer 48 has already exist!
```

3) 可输入“remove peer x”来将结点移出网络

```
remove peer 1
remove OK!
```

```
I have been removed from the network!
```

若出现下图的情况，说明网络中不存在该结点，移出操作失败：

```
Traceback (most recent call last):
  File "cdht.py", line 306, in <module>
    main()
  File "cdht.py", line 301, in main
    find_peer(old_peer, "remove", peer)
  File "cdht.py", line 215, in find_peer
    send_tcp_message(message, address)
  File "cdht.py", line 87, in send_tcp_message
    tcp.connect(address)
ConnectionRefusedError: [WinError 10061] 由于目标计算机积极拒绝，无法连接。
```

4) 可输入“ctrl + c”或点击关闭按钮强行关闭结点。网络如果未收到 2 个及以上 ping response 信息，就会发现该结点离开,并提醒相关结点更换后继。

```
peer 25 is not alive.
My first successor is now 30.
A ping response was received from peer 30
My second successor is now 31.
```

5) 如果出现下图的情况，说明您的计算机的某个程序占用了该结点的端口号，因此该结点初始化失败：

```
D:\python\CircularDHT\CircularDHT>python cdht.py 50 1 3
Initialised Peer 50. Successor first is 1, successor second is 3
Please input 'insert peer i' or 'remove peer i':
Exception in thread Thread-2:
Traceback (most recent call last):
  File "D:\Anaconda3\lib\threading.py", line 926, in _bootstrap_inner
    self.run()
  File "D:\Anaconda3\lib\threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "cdht.py", line 59, in set_udp_port
    udp.bind(address)
OSError: [WinError 10013] 以一种访问权限不允许的方式做了一个访问套接字的尝试。
```