目录

[1. 绪论 3](#_Toc75110707)

[2. 设计主体 3](#_Toc75110708)

[2.1 RIP工作原理 3](#_Toc75110709)

[2.1.1 Bellman-Ford算法 3](#_Toc75110710)

[2.1.2向量表更新 4](#_Toc75110711)

[2.1.3 路由查找 5](#_Toc75110712)

[2.2 课程设计具体思路 5](#_Toc75110713)

[2.2.1 建立模型 5](#_Toc75110714)

[2.2.2 选取模型数据 7](#_Toc75110715)

[2.2.3 构建数据库 7](#_Toc75110716)

[2.2.4 功能设计 8](#_Toc75110717)

[2.2.5 界面设计 8](#_Toc75110718)

[2.3功能实现 10](#_Toc75110719)

[2.3.1 实现网络拓扑结构 10](#_Toc75110720)

[2.3.2 构建路由表 11](#_Toc75110721)

[2.3.3 查找路由 12](#_Toc75110722)

[3. 结束语 13](#_Toc75110723)

[4. 参考资料 14](#_Toc75110724)

1. 绪论

互联网是由不同路由器连接的网络组成的。当数据报文从源点出发到目的地的过程中，需要通过很多路由器，直到连接着目的地址的路由器才会停下，在同一个自治系统（Autonomous system）中一般使用一致的路由选择协议，理解路由选择协议便是理解互联网的连接世界的基础。本课设中使用的路由选择协议为基于距离向量算法的RIP（Routing Information Protocol），即路由信息协议，接下来本文讲讨论RIP工作原理、课程设计的具体思路、以及实际操作过程对算法的改动。

2. 设计主体

2.1 RIP工作原理

2.1.1 Bellman-Ford算法

路由信息协议基于距离向量算法，使用这种算法类型的路由被称之为“距离向量路由”。

距离向量路由中，自治系统内所有的路由器和网络都表示为一个图。路由器通常可以用一个节点表示，一个网络用连接两个节点的链接表示，尽管其他表示也是可行的。这里使用了一种被称为 Bellman-Ford（也称为 Ford-Fulkerson）的算法，来在一段时间内找到最短路径给定节点之间的距离。

Bellman-Ford算法可以用来解决很多图论相关的问题，如果我们知道每对结点之间的距离，我们就可以使用这个算法求得任意两个结点之间的最短距离。如图2.1，Bellman-Ford算法可以求得任意一个结点到其他结点的最短距离，而对所有结点进行单源最短路径求解后，便可以获得所有结点到其他结点的最短距离。

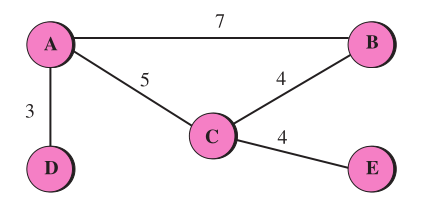


图2.1 Bellman-Ford算法例图

算法的主要思想是基于所有的i邻结点都知道到达目的结点j的最短距离，所以结点i与j之间的最短距离可以通过将i到所有邻结点x的距离（包括i）D(i,x)加上x结点到j结点的最短距离d(x,j)动态获得，即

通过迭代的方式来构建每个结点的最短路径：

1. 将每个结点到其自身的最短距离设置为0
2. 将每个结点到其他结点的最短距离设置为正无穷，同时应该给出每个结点到其他结点的距离（如果没有连接则视为正无穷）
3. 算法不断重复直到没有最短距离发生改变

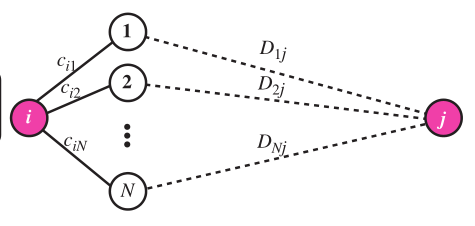


图2.2 Bellman-Ford算法思想

2.1.2向量表更新

距离向量路由借助Bellman-Ford算法思想，动态地更新路由表。路由表记录着从该路由节点到目的网络的距离以及下一步需要跳往哪个路由或者接口。路由过程中，每隔一段时间，路由器便会向相邻的路由器发送自身路由表信息，接收者通过接收到的路由表，来动态地更新自身的路由表。这符合Bellman-Ford的核心思想。例如，路由器R1到网络N3的距离为10，此时路由器R2向R1发来路由信息，R2到N3的距离为5，那么R1则更新路由表的相关条目，将前往N3的距离更新为5+1=6，将该条目的下一条改为R2。

距离向量路由在工作时遵循如下原则：

1. 路由器之间交换的信息被成为RIP报文
2. 仅相邻的路由器之间能够交换信息
3. 路由器交换的信息是当前路由器知道的全部信息
4. 每个路由器应该保存至少三条信息：目的网络N、距离cost、下一跳next\_hop
5. 对于使用划分子网或者无分类编址的网络，应该增加“掩码”信息

更新路由表的具体过程如下：

1. 当路由器接收到相邻路由表的发来的RIP报文，先修改此报文中所有的项目，把“下一跳”字段中的地址都改为相邻路由器的地址X，并把所有的“距离”字段的值加1
2. 对修改后的RIP报文中的每一个条目进行如下操作
   1. 如果原来的路由表中没有目的网络为N的条目，则添加入路由表
   2. 如果原路由表中存在目的网络为N的条目
      1. 如果该条目的“下一跳”为发送报文的相邻路由器的地址，则直接更新该条目
      2. 如果存在目的网络为N但是“下一条”不为X的条目，则比较收到的条目中的“距离”与当前条目的“距离”值，如果小于，则替换整个条目
   3. 如果三分钟没有收到相邻路由表的更新报文，则将此相邻路由表设为不可达
3. 返回

2.1.3 路由查找

当IP报文到达的时候，下一步的转发方向可以通过查找路由表来判断，具体过程如下：

1. 从收到的数据报的首部提取目的IP地址D。
2. 先判断是否为直接交付。对路由器直连的网络逐个进行检查，用各网络的子网掩码和D逐位相与，看结果是否和相应的网络地址匹配，若匹配则把分组直接交付，否则是间接交付，执行3。
3. 若路由表中有目的地址为D的特定主机路由，则把数据报传送给路由表中指定的下一条路由，否则执行4。
4. 对表中的每一行，用其中的子网掩码和D逐位相与，其结果为N。若N与目的网络地址匹配，则转发给“下一跳”中指定的路由，否则执行5。
5. 若路由表中有一个默认路由，则把数据报传送给路由表中所指明的默认路由。
6. 报告转发出现错误。

2.2 课程设计具体思路

2.2.1 建立模型

将网络中的结点抽象成Router（路由器）和Net（网络），Router，结点之间的边抽象为Arc，路Router由id属性唯一确定（对于本课程设计来说name也可以唯一确定router）；net属性即CIDR记法的网络表示来唯一确定一个Net；每对结点之间的连接Arc包含两端结点的网络地址，属性前的from\_以及to\_前缀仅用来在构建路由表时，表示模型的遍历方向，整体的图存储模型采用了“邻接表”。Arc本身是属于无向图的边，而各个属性的用途将在下一小节说明。

**class** **Router**():

**def** \_\_init\_\_(self, id, router\_name, router\_dic):

self.id = id

self.name = router\_name

self.router\_dic = router\_dic

self.next = **None***##list of class Arc*

**class** **Net**():

**def** \_\_init\_\_(self, net, net\_name):

self.net = net

self.mask = get\_mask(net)

self.name = net\_name

self.next = **None***##list of class Arc*

self.from\_router = **None**

self.from\_ip = **None**

**class** **Arc**():

**def** \_\_init\_\_(self, from\_type, to\_type, from\_net, to\_net):

self.from\_type = from\_type

self.to\_type = to\_type

self.from\_net = from\_net

self.to\_net = to\_net

self.next = **None**

self.from\_router = -1*#默认是-1*

self.to\_router = -1*#默认是-1*

代码2.1

将路由表的每一项抽象为TableEntry，基本属性已在2.1节说明。

**class** **TableEntry**():

**def** \_\_init\_\_(self, router\_id, r\_name, des, sub\_mask, cost, next\_hop, next\_ip):

self.id = router\_id#向量表所在路由表的id

self.r\_name = r\_name#向量表所在路由表的名字

self.des = des#目的网络

self.mask = sub\_mask#子网掩码

self.cost = cost#距离

self.next\_id = next\_hop#下一跳的路由器或者接口名

self.next\_ip = next\_ip#下一跳ip

代码2.2

2.2.2 选取模型数据

这里采用TCP/IP Protocol Suite书内的实例

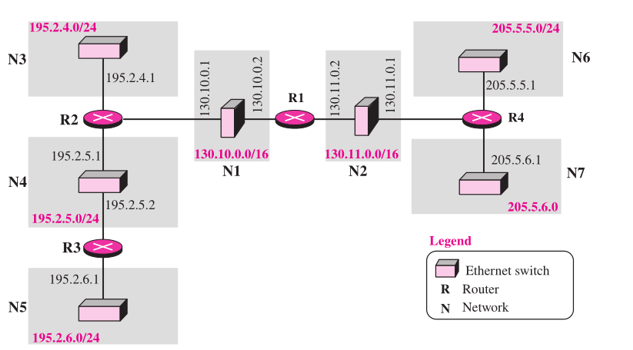


图2.3 选取网络拓扑模型数据

2.2.3 构建数据库



图2.3 数据库ER图模型

图表

描述已自动生成

图2.4 抽象拓扑图

图形用户界面, 文本, 表格

描述已自动生成

图2.5 数据库net表

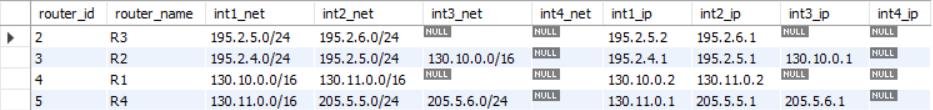


图2.6 数据库router表

2.2.4 功能设计

图形用户界面, 日程表

描述已自动生成

图2.7 功能设计流程图

2.2.5 界面设计

图形用户界面

中度可信度描述已自动生成图形用户界面, 应用程序

描述已自动生成

表格

描述已自动生成图形用户界面

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

图2.8 界面设计

2.3功能实现

2.3.1 实现网络拓扑结构

从数据库中读取所有net和router数据，分别存入字典net\_dic和router\_dic中，net\_dic以CIDR记法的前缀和网络号为键值，router\_dic以路由器的id为键值。

同时构建邻接表，遍历所有的路由器Router，并对每个Router四个接口进行遍历，若存在网络连接，则构建一个边Arc，设置相关参数。由于图为无向图，可以利用接口上记录的网络号同时在字典中找到对应的Net类，再次设置Arc。

日程表

中度可信度描述已自动生成

图3.1 数据结构示例

**def** build\_nodes\_list(routers, nets):

routerNodes = []

**for** router **in** routers:

rNode = Router(router['router\_id'], router['router\_name'], router)

routerNodes.append(rNode)

net\_dic = {}

**for** net **in** nets:

nNode = Net(net['net\_ip\_mask'], net['net\_name'])

net\_dic[net['net\_ip\_mask']] = nNode

router\_dic = {}

**for** node **in** routerNodes:

router\_dic[node.id] = node

temp\_node = node

r = node.router\_dic

**for** i **in** range(4): *##遍历四个接口*

int\_net = 'int' + str(i + 1) + '\_net'

int\_ip = 'int' + str(i + 1) + '\_ip'

**if** r[int\_net] **is** **not** **None** **and** r[int\_ip] **is** **not** **None**:

net\_node = net\_dic[r[int\_net]]

arc = Arc(NodeType.ROUTER, NodeType.NET, r[int\_ip], r[int\_net])

arc.set\_from\_router\_id(node.id)

temp\_node.next = arc

temp\_node = temp\_node.next

**while** net\_node.next **is** **not** **None**:

net\_node = net\_node.next

arc = Arc(NodeType.NET, NodeType.ROUTER, r[int\_net], r[int\_ip])

arc.set\_to\_router\_id(node.id) *# 设置目的路由器id，路由器有多个接口，只能通过id才能快速找到（其实通过数据库查找也可以）*

net\_node.next = arc

**return** (router\_dic, net\_dic)

2.3.2 构建路由表

本课设针对的是路由查找的模拟，上一节构建的拓扑图中所有的边的距离相同，默认为1，故在构建路由表的时候采用了更加简单的BFS单源路径求最短距离算法。

首先从每个网络顶点出发，进行BFS遍历，设置cost\_to\_net，记录到每个网络顶点的最短距离，cost\_to\_router记录到每个路由表的最短距离，由于拓扑图的特殊性，当我们遍历到某个路由结点的时候，上一个结点一定是一个网络结点，所以我们每个路由结点到起始网络结点的最短距离与前一个网络结点到起始网络节点的最短距离相等，当我们通过遍历网络结点的Arc链表从而遍历到某个路由结点的时候，路由器到起始网络结点的最短距离便等于该网络结点到起始结点的最短距离。

最大的难点在于记录下一跳，这里使用了一个简便的方式，即当我们从一个路由结点遍历到一个网络结点N的时候，便将路由结点以及其和该网络结点连接处的ip地址记录进N中，这样当以该网络结点开始进行下一层的BFS的时候，遍历到的路由便可以从N中获得前一个路由的信息和下一跳的ip。在2.2节的模型构建中便加入了这两个属性：

**class** **Net**():

**def** \_\_init\_\_(self, net, net\_name):

self.net = net

self.mask = get\_mask(net)

self.name = net\_name

self.next = **None***##list of class Arc*

self.from\_router = **None#记录前一个路由**

self.from\_ip = **None#记录连接ip**

**def** bfs\_min\_distance(router\_dic, net\_dic):

*'''*

*soul of the whole code*

*:param router\_dic:*

*:param net\_dic:*

*:return:*

*'''*

. . . #省略了变量定义

**while** **not** t\_queue.empty():

node = t\_queue.get()

arc = node.next

**while** arc **is** **not** **None**:

**if** (arc.to\_type **is** NodeType.NET **and** **not** net\_visited[arc.to\_net]):

net\_visited[arc.to\_net] = **True**

t\_queue.put(net\_dic[arc.to\_net])

cost\_to\_net[arc.to\_net] = cost\_to\_router[arc.from\_router] + 1

net\_dic[arc.to\_net].set\_from\_router(node, arc.from\_net)

**elif** (arc.to\_type **is** NodeType.ROUTER **and** **not** router\_visited[arc.to\_router]):

router\_visited[arc.to\_router] = **True**

t\_queue.put(router\_dic[arc.to\_router])

cost\_to\_router[arc.to\_router] = cost\_to\_net[arc.from\_net]

*# pre = None if pre\_router\_queue.empty() else pre\_router\_queue.get()*

pre\_r = **None** **if** net\_dic[arc.from\_net] **is** **None** **else** net\_dic[arc.from\_net].from\_router

**if** pre\_r **is** **not** **None** **and** pre\_r.id **is** router\_dic[arc.to\_router].id:

pre\_r = **None**

pre\_i = **None** **if** pre\_r **is** **None** **else** net\_dic[arc.from\_net].from\_ip

entry = TableEntry(router\_id=arc.to\_router,

des=net, sub\_mask=get\_mask(net),

cost=cost\_to\_router[arc.to\_router],

next\_hop=**None** **if** pre\_r **is** **None** **else** pre\_r.id,

r\_name=router\_dic[arc.to\_router].name,

next\_ip=pre\_i)

tables.append(entry)

arc = arc.next

**return** tables

2.3.3 查找路由

接收输入目的ip以及选择路由器后，便从数据库中获得该路由器的路由表，遍历路由表，并将目的地ip与路由表的掩码进行与操作，如果结果与路由表中的目的网络一致，则视为查找成功，否则查找失败。

**def** start\_test():

router = box.get()

des = ip\_input.get()

**if** router **is** **not** **None**:

table = sql\_get\_router\_table\_by\_name(router, cur, db)

print(table)

**if** des **is** **not** **None**:

print("正在遍历路由表...")

**for** entry **in** table:

print("当前条目：目的网络:" + entry['des\_net'] + ', 掩码:'+entry['mask\_code'] + '下一跳地址:' + entry['next\_ip'])

**if** check\_is\_des(entry['des\_net'], des, entry['mask\_code']):

print("查找成功")

print(entry['nr\_name'])

print(entry['next\_ip'])

**break**

print('查找失败')

**else**:

print('输入目的网络')

**else**:

print("选择路由器")

文本

描述已自动生成

图3.2 查找结果

3. 结束语

通过本次课程设计让我对路由表的工作原理有了进一步的认识，并且在此期间提高了对python的掌握程度，并且对数据库的操作也有了进一步的了解。但是代码不足的地方还有很多，比如没有使用Bellman-Ford算法，以及没有加入默认路由和指定路由等，希望接下来有时间的时候能够继续完善，虽然只是个模拟项目，但希望能让它更真实一些。

4. 参考资料

[1] 谢希仁.计算机网络[M].北京:电子工业出版社, 2007.1

[2] Behrouz A. Forouzan. TCP/IP protocol suite[M].New York: McGraw-Hill,2010

[3] Will-kkc.正则表达式之子网掩码mask.[EB/OL].https://blog.csdn.net/u012219045/article/details/98957753