Dijkstra算法模拟

路由器协议实践报告

分工**：**

白雪：完成拓扑图、组稿

陈占貂：系统测试、撰写测试报告

崔宪强：系统功能及系统设计

贾大力：目标和算法原理简介

常立凡、陈瑞：系统实现

# 研究目标

1、模拟路由器协议的过程，形成多个节点的网状拓扑，每个节点拥有一个路由表、以及相关信息状态，设置各条边的距离值，画出拓扑图。

2、编写程序自动完成模拟过程，采用离散时间单位，可视化，可交互，人工操作的单步前进、多步快进，最后生成日志、结果（任意两点间的路径）。

3、每一步展示：节点之间发送的距离矢量信息、各节点更新后的路由表内容。测试中途删除某个节点，继续模拟协议运行过程，并得出正确的结果。

# 二.算法原理简介

## 2.1迪杰斯特拉(Dijkstra)算法基本思想

1、通过Dijkstra计算图G中的最短路径时，需要指定起点s(即从顶点s开始计算)。

2、引进两个集合S和U。S的作用是记录已求出最短路径的顶点(以及相应的最短路径长度)，而U则是记录还未求出最短路径的顶点(以及该顶点到起点s的距离)。

3、初始时，S中只有起点s；U中是除s之外的顶点，并且U中顶点的路径是”起点s到该顶点的路径”。然后，从U中找出路径最短的顶点，并将其加入到S中；接着，更新U中的顶点和顶点对应的路径。 然后，再从U中找出路径最短的顶点，并将其加入到S中；接着，更新U中的顶点和顶点对应的路径。 … 重复该操作，直到遍历完所有顶点。

## 2.2Dijkstra算法具体计算步骤

1、初始时，S只包含起点s；U包含除s外的其他顶点，且U中顶点的距离为”起点s到该顶点的距离”[例如，U中顶点v的距离为(s,v)的长度，然后s和v不相邻，则v的距离为∞]。

2、从U中选出”距离最短的顶点k”，并将顶点k加入到S中；同时，从U中移除顶点k。

更新U中各个顶点到起点s的距离。之所以更新U中顶点的距离，是由于上一步中确定了k是求出最短路径的顶点，从而可以利用k来更新其它顶点的距离；例如，(s,v)的距离可能大于(s,k)+(k,v)的距离。

3、重复步骤(2)和(3)，直到遍历完所有顶点。

# 三.系统功能

路由协议模拟系统主要根据Dijkstra算法找出路由拓扑中任意两点之间的最短路径,并在每次增加一个新的相邻路由节点时对路由表进行更新，同时打印出每一次路由更新的log。

# 四.系统设计

## 4.1模块设计：

**DVdata\_int\_data**：实现路由表的打印。

**Data\_print**:实现路由表的更新，包括节点的增加以及各个节点下最短路径的更新。

**fun\_update**:实现节点的遍历。

**node\_update**:更新最短路径及下一路由。

## 4.2网络拓扑图设计

网络拓扑图设计为9个路由器route0-route8，各路由器网络带宽通过权重表示。拓扑图如下图所示:

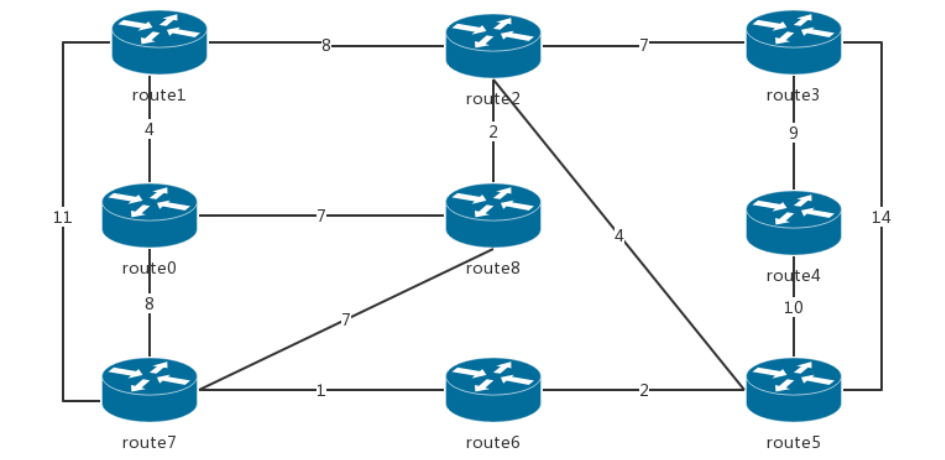


图-1拓扑图

## 4.3时序图：

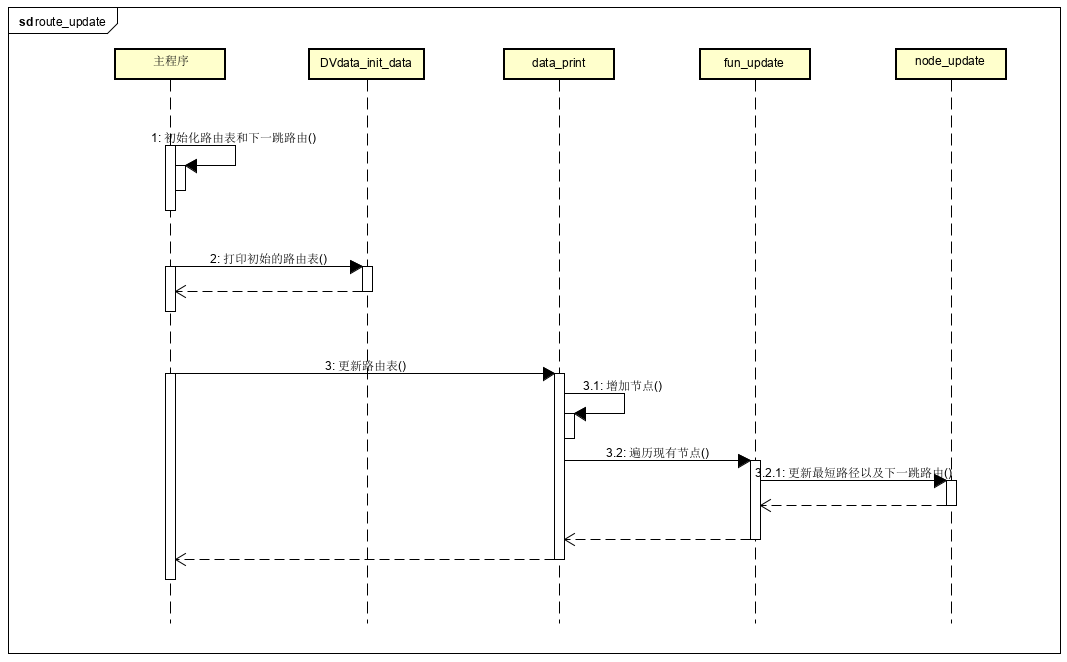
****

图-2时序图

# 五.系统实现

## 5.1源代码

KEYDATA\_ZERO = 0

KEYDATA\_FIRST = 1

KEYDATA\_SECOND = 2

KEYDATA\_THIRD = 3

KEYDATA\_FOURTH = 4

KEYDATA\_FIFTH = 5

KEYDATA\_SIXTH = 6

KEYDATA\_EVENTH = 7

KEYDATA\_EIGHTH = 8

KEYDATA\_NINTH = 9

KEYDATA\_TENTH = 10

KEYDATA\_ELEVENTH = 11

KEYDATA\_FOURTEENTH = 14

def set\_link(i, j, value):

NODE\_V[i][j] = value

NODE\_V[j][i] = value

def select\_data(cans):

data = ""

if (cans == 'inf'):

data = "no road access"

else:

data = cans

return data

def node\_update(i, j, NODE\_V, nd\_link, NODE\_HOP):

list = []

flag = -1

length\_for\_list = 0

list\_hop\_router = []

for k in range(KEYDATA\_NINTH):

for adj\_node in nd\_link[j]:

if NODE\_V[j][adj\_node]+NODE\_V[adj\_node][k] < NODE\_V[j][k]:

NODE\_V[j][k] = NODE\_V[j][adj\_node]+NODE\_V[adj\_node][k]

NODE\_HOP[j][k] = adj\_node

if (flag != j):

list = []

length\_for\_list = len(list)- 1

list.append(k-1)

list\_hop\_router.append(adj\_node)

'''print("adj\_node is : ",adj\_node)

print("k is : ",k)

print("j: ", j)

print("append data is : ",NODE\_V[adj\_node][k])'''

else:

list\_hop\_router.append(-1)

if (len(list) > length\_for\_list):

*#print("i: ", j, ", j: ", k)*

if (flag != j):

list.insert(0,j)

list.insert(len(list),k)

flag = j

else:

list.insert(len(list),k)

*#print(list)*

*#print()*

length\_for\_list = len(list)

*#print('Table after NODE\_V algorithm:')*

for m in range(k):

*#print("i: ", j, ", j: ", k)*

print(" ", "every line to update route: ")

print(" ", "Purpose of the network ", "distance ", "Next hop router")

for j in range(len(NODE\_V[m])):

if m != j:

print(" ", 'N%d'%(j), " ", NODE\_V[m][j]," ", "N",NODE\_HOP[m][j])

print()

def fun\_update(NODE\_V, nd\_link, NODE\_HOP):

print()

for i in range(10):

for j in range(len(nd\_link)):

node\_update(i, j, NODE\_V, nd\_link , NODE\_HOP)

*# print(f'the {i+1} time update', NODE\_V)*

print('Table after NODE\_V algorithm:')

for i in range(len(NODE\_V)):

print("Router ", i, " updated routing table is :")

print("Purpose of the network ", "distance ", "Next hop router")

for j in range(len(NODE\_V[i])):

if i != j:

print('N%d'%(j), " ", NODE\_V[i][j]," ", "N",NODE\_HOP[i][j])

*#print('N%d(0,1,2,...,9)=%s'%(i,NODE\_V[i]))*

print()

def data\_print(NODE\_V, nd\_link, NODE\_HOP):

print("Add nodes for the first time, we don't add any note : ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_FIRST,KEYDATA\_EVENTH])

print("Add nodes for the second time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_ZERO,KEYDATA\_SECOND,KEYDATA\_EVENTH])

print("Add nodes for the third time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_FIRST,KEYDATA\_THIRD,KEYDATA\_FIFTH,KEYDATA\_EIGHTH])

print("Add nodes for the fourth time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_SECOND,KEYDATA\_FOURTH,KEYDATA\_FIFTH])

print("Add nodes for the fifth time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_THIRD,KEYDATA\_FIFTH])

print("Add nodes for the sixth time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_SECOND,KEYDATA\_THIRD,KEYDATA\_FOURTH,KEYDATA\_SIXTH])

print("Add nodes for the seventh time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_FIFTH,KEYDATA\_EVENTH,KEYDATA\_EIGHTH])

print("Add nodes for the eighth time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_ZERO,KEYDATA\_FIRST,KEYDATA\_SIXTH,KEYDATA\_EIGHTH])

print("Add nodes for the ninth time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

nd\_link.append([KEYDATA\_SECOND,KEYDATA\_SIXTH,KEYDATA\_EVENTH])

print("Add nodes for the last time: ")

fun\_update(NODE\_V, nd\_link, NODE\_HOP);

def linkdata\_pad():

set\_link(KEYDATA\_ZERO,KEYDATA\_FIRST,KEYDATA\_THIRD)

set\_link(KEYDATA\_ZERO,KEYDATA\_EVENTH,KEYDATA\_EIGHTH)

set\_link(KEYDATA\_FIRST,KEYDATA\_SECOND,KEYDATA\_EIGHTH)

set\_link(KEYDATA\_FIRST,KEYDATA\_EVENTH,KEYDATA\_ELEVENTH)

set\_link(KEYDATA\_SECOND,KEYDATA\_THIRD,KEYDATA\_EVENTH)

set\_link(KEYDATA\_SECOND,KEYDATA\_FIFTH,KEYDATA\_THIRD)

set\_link(KEYDATA\_SECOND,KEYDATA\_EIGHTH,KEYDATA\_SECOND)

set\_link(KEYDATA\_THIRD,KEYDATA\_FOURTH,KEYDATA\_NINTH)

set\_link(KEYDATA\_THIRD,KEYDATA\_FIFTH,KEYDATA\_FOURTEENTH)

set\_link(KEYDATA\_FOURTH,KEYDATA\_FIFTH,KEYDATA\_TENTH)

set\_link(KEYDATA\_FIFTH,KEYDATA\_SIXTH,KEYDATA\_SECOND)

set\_link(KEYDATA\_SIXTH,KEYDATA\_EVENTH,KEYDATA\_FIRST)

set\_link(KEYDATA\_SIXTH,KEYDATA\_EIGHTH,KEYDATA\_SIXTH)

set\_link(KEYDATA\_EVENTH,KEYDATA\_EIGHTH,KEYDATA\_EVENTH)

set\_link(KEYDATA\_ZERO,KEYDATA\_EIGHTH,KEYDATA\_EVENTH)

def DVdata\_init\_data(NODE\_V, NODE\_HOP):

print('Initialize table:')

for i in range(len(NODE\_V)):

print("Router ", i, " updated routing table is :")

print("Purpose of the network ", "distance ", "Next hop router")

for j in range(len(NODE\_V[i])):

if i != j:

print('N%d'%(j), " ", NODE\_V[i][j], " ", "N",NODE\_HOP[i][j])

print('')

NODE\_V = []

NODE\_HOP = []

if \_\_name\_\_ == '\_\_main\_\_':

NODE\_V =[[float('inf') for i in range(KEYDATA\_NINTH)] for i in range(KEYDATA\_NINTH)]

for i in range(KEYDATA\_NINTH):

NODE\_V[i][i] = 0

NODE\_HOP =[[float('inf') for i in range(KEYDATA\_NINTH)] for i in range(KEYDATA\_NINTH)]

for i in range(KEYDATA\_NINTH):

NODE\_HOP[i][i] = -1

linkdata\_pad()

DVdata\_init\_data(NODE\_V, NODE\_HOP)

nd\_link = []

data\_print(NODE\_V, nd\_link, NODE\_HOP)

# 六.系统测试

本次测试在python环境下，通过对比测试用例输出的路由信息来判断每台路由器的路由表的更新是否正确。

## 6.1测试步骤

主要测试步骤：

1.测试开始路由器连接时候的路由状况。

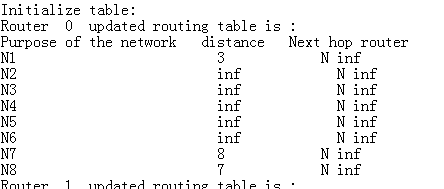
2.测试依次增加路由表之后，原有路由器的路由表更新状况，以及新加路由器的路由表状况，并能从中找到一条最佳的路由路径。

3.能够在全部路由器加入的情况下，通过最佳路由路径从各个路由器间传递信号。

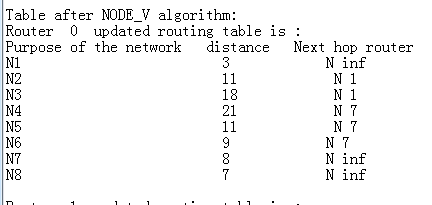
4.当其中一个路由器发生故障时，其他路由器能够获悉，并从中改变各自路由器中的路由表状况。

## 6.2测试结果

简单列出编号为0的路由器在连接一次的路由表更新状况



简单列出编号为2的路由器断开与编号8路由器连接的编号0的路由表结果



以上结果完全符合我们的预期结果，初步测试通过。（详细测试输出见附件）

# 七.总结

通过这门课程的学习，我们已经初步了解了计算机网络知识，但那些毕竟还是属于理论，所以，在我们做了这个课程设计，求最短路径是数据结构的基础中比较重要内容，掌握这类知识对我们得学习很有帮助。通过这次课程设计我们要掌握Dijkstra算法思想、掌握Floyd算法思想、掌握Dijkstra、了解Prim Kruskal算法。通过这次课程设计，我们对上课所学得知识进行了巩固，对最小生成树、最短路径、Dijkstra算法，最短路由有了更深得理解。在课程设计的过程中我也遇到了许多得困难，例如对最短路径在程序上实现的方式等。

在整个的制作过程中，编写程序和了解算法原理是最困难的，由于对这些得不熟悉，花费了很多得时间，通过和同学去图书馆查询，终于找到了解决的办法。学好这门课程对我们在以后得通信学习有着很大得帮助。在我们遇到困难的时候，是老师和同学极力帮助，知道我们解决遇到的问题，非常感谢他们的支持与帮助。

短暂的两个星期过去了，我们完成了高级计算机网络的课程设计，虽然在这过程中遇到了许多得困难，但成功还是给我们带来了更大的喜悦。相信我们的网络学习之路不会就此结束，我们会更加努力地学习！

# 附件：

源代码、测试输出、完整log。