计算机网络编程
实验报告

班级：07111707

组长：1120171189 崔程远

成员：1120172149 吴沁璇

1120172153 张澈

1120172163 王晓媛

1120172736 张鉴昊

1120172765 曾煜瑾

1120173326 曾紫飞

北京理工大学

计算机学院

2020 年 5 月

第五章 实验 2　链路状态路由（LS）算法

1. 实验目的

模拟网络层中根据网络结构自适应调整路由表的状态路由算法，加深对 LS 算法的理解和掌握。

2. 实验内容

编写一个基于链路状态的路由算法，生成路由表。

配置文件关键要点：

配置文件中存放一个二维矩阵，表示网络拓扑中各个路由结点与其它结点之间的邻接关系。若两个结点直接相邻，则对应矩阵元素的值为两个结点之间的距离（正整数）。若两个结点之间不直接相邻，则对应矩阵元素的值为空（或 99，表示无穷远）。

例如：

```
//配置文件内容示例，共计 5 个结点，99 代表不可达
0 7 99 99 10
7 0 1 99 8
99 1 0 2 99
99 99 2 0 2
10 8 99 2 0
```

程序运行屏幕输出要点：

运行程序，输出初始网络拓扑（配置文件中）对应的各个路由器的链路状态，之后根据最短路径优先 SPF 算法计算任意路由器之间的最短路径，输出每个路由器每步的最短通路结果，再输出最终态各个路由器的路由表。

3. 实验原理

一个主机通常与一台路由器相连接，该路由器即为主机的默认路由器。源主机的默认路由器称作源路由器，目的主机的默认路由器称作目的路由器。一个分组从源主机到目的主机的路由选择问题即从源路由器到目的路由器的路由选择问题。

路由选择算法可分为：全局式路由选择算法和分散式路由选择算法；

全局式路由选择算法：所有路由器掌握完整的网络拓扑和链路费用信息，例如链路状态（ls）路由算法

分散式路由选择算法：路由器只掌握物理相连的邻居以及链路费用，例如距离向量（dv）路由算法

下面主要介绍链路状态路由算法（Link-Stage）中的一种典型算法 OSPF 算法。

一、OSPF

Open Shortest Path First，开放最短路径优先协议，是一种开源的使用最短路径优先（SPF）算法的内部网关协议（IGP）。常用于路由器的动态选路。

二、OSPF 常见概念

a) 邻居：宣告 OSPF 的路由器从所有启动 OSPF 协议的接口上发出 hello 数据包。如果两台路由器位于同一条数据链路上，并且它们根据互相的 hello 消息中指定的某些信息（比如 id 等）协商成功，那么它们就成为了邻居（Neighbor）。

b) 邻接关系（Adjacency）：两台邻居路由器之间构成的一条点到点的虚链路，邻接关系的建立是由交换 Hello 信息的路由器类型和网络类型决定的。

c) 链路状态通告（Link State Advertisement，LSA）：每一台路由器都会在所有形成邻接关系的邻居之间发送链路状态通告 LSA。LSA 描述了路由器所有的链路、接口、邻居等信息。OSPF 定义了许多不同的 LSA 类型。

d) 链路状态数据库（LSDB）：每一台收到来自邻居路由器发出的 LSA 的路由器都会把这些 LSA 信息记录在它的 LSDB 中，并且发送一份 LSA 的拷贝给该路由器的其他所有邻居。这样当 LSA 传播到整个区域后，区域内所有的路由器都会形成同样的 LSDB。

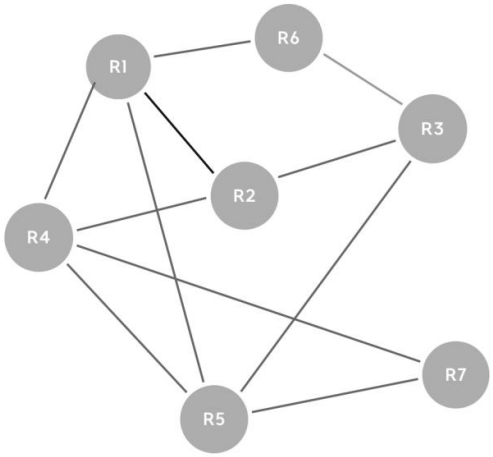
三、OSPF 基本原理

a) OSPF 算法是让每个路由器中的数据库储存整个网络的拓扑图，即每个路由器掌握了全局的信息，此时这个网络趋于稳定，便可以使用单源最短路径（Dijkstra）来选择路由。

b) 每个路由器与其邻居的通信行为有以下几种：

i. 保持联系：整个自治系统中，每个路由器都有唯一标识 routerid(32-bit). 与其每个邻居间隔 30s，发送一次 hello 报文，确认对方是否活跃；二者相互通信发送 hello，并收到对方回应 hello。双方会周期性将自己的路由数据摘要发送给对方，一般 30min

ii. 告知现今情况



如图，R1 会周期性地将自己的路由摘要发送给所有邻居。比如对 R6 路由器，R1 会发送称为 DD 报文的包，里面会说自己认识 R6/R2/R4/R5, R6 对比自己的信息库发现没有除自己之外的其他路由信息，于是就发送请求报文，请求告知这些陌生路由详情。这个请求报文称为 LSR(链路状态请求报文)报文。R1 收到之后，发送 LSU 报文（链路状态更新）告知 R6 详情。R6 收到之后，给 R1 个确认-LSACK 报文。此时这两个路由器的信息库一致，即全毗邻关系。

4. 实验环境

语言	集成开发环境	编译器
C++	Visual Studio 2017	gcc version 4.8.1
Java	Eclipse 2019	java version "1.8.0_65"
Python	Pycharm 2017	Python 3.7.0

5. 实验步骤

三份代码的结构和输出完全一致，均采用面向对象的构造方法，下面以 C++ 代码为例进行分析。

• 全局变量

```
private:
    int adjMat[100][100] = {
                { 0, 7, 99, 99, 10 },
                { 7, 0, 1, 99, 8 },
                { 99, 1, 0, 2, 99 },
                { 99, 99, 2, 0, 2 },
                { 10, 8, 99, 2, 0 } };
    int distance[n][n];
    int nextRouter[n][n];
    string s = "ABCDEFGHIJKLMN";
```

• 根据邻接矩阵初始化 distance 矩阵和 nextRouter 矩阵

```
void Initialize()
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                distance[i][j] = adjMat[i][j];
                nextRouter[i][j] = j;
            }
        }
    }
```

• 输出初始网络拓扑对应的各个路由器的链路状态

```
void printInitialLinkState()
    {
        cout << "Initial Link State For Each Router is:" << endl;
        for (int i = 0; i < n; i++)
        {
            cout << "router " << s[i] << endl;
            for (int j = 0; j < n; j++)
            {
                if (adjMat[i][j] != 0 && adjMat[i][j] != 99) //不是本节点且相邻
                {
                    cout << s[j] << " " << adjMat[i][j] << endl;
                }
            }
            cout << endl;
        }
    }
```

• Djikstra 算法计算任意路由器之间的最短路，并输出每个路由器每步的最短通路结果

```
void SPF(int source)
    {
        cout << "For Router " << s[source] << endl;
        int flag[n];
        for (int j = 0; j < n; j++)
        {
            flag[j] = 0;
        }
        flag[source] = 1;
        for (int i = 0; i < n; i++)
        {
            int k = source;
            int min = 99;
            for (int j = 0; j < n; j++)
            {
                if (distance[source][j] < min && flag[j] == 0)
                {
                    min = distance[source][j];
                    k = j;
                }
            }
            if (k == source)
            {
                cout << endl;
                return;
            }
            cout << "Step " << i << ": " << s[source];
            int next = source;
            while (nextRouter[next][k] != k)
            {
                cout << "-" << s[nextRouter[next][k]];
                next = nextRouter[next][k];
            }
            cout << "-" << s[k] << " " << min << endl;
            flag[k] = 1;
            for (int j = 0; j < n; j++)
            {
                if (distance[source][k] + adjMat[k][j] < distance[source][j] &&
flag[j] == 0)
                {
                    distance[source][j] = distance[source][k] + adjMat[k][j];
                    int next2 = source;
```

```
                    while (next2 != k)
                    {
                        nextRouter[next2][j] = nextRouter[next2][k];
                        next2 = nextRouter[next2][k];
                    }
                }
            }

        }

    }
```

·输出最终态各个路由器的路由表

```
void printFinalRoutingTable()
    {
        cout << "Final Routing Table For Each Router is:" << endl;
        for (int i = 0; i < n; i++)
        {
            cout << "Router " << s[i] << endl;
            for (int j = 0; j < n; j++)
            {
                cout << s[nextRouter[i][j]] << " " << distance[i][j] << endl;
            }
            cout << endl;
        }
    }
```

·主函数

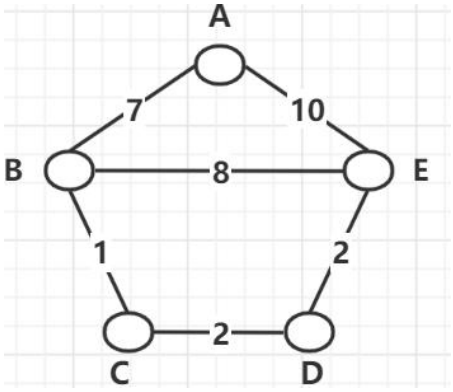
```
int main()
{
    LS operation;
    operation.Initialize();
    operation.printInitialLinkState();
    cout << "Shortest Way Of Each Step For Each Router is:" << endl;
    for (int i = 0; i < n; i++)
    {
        operation.SPF(i);
    }
    operation.printFinalRoutingTable();
    system("pause");
}
```

6. 实验结果

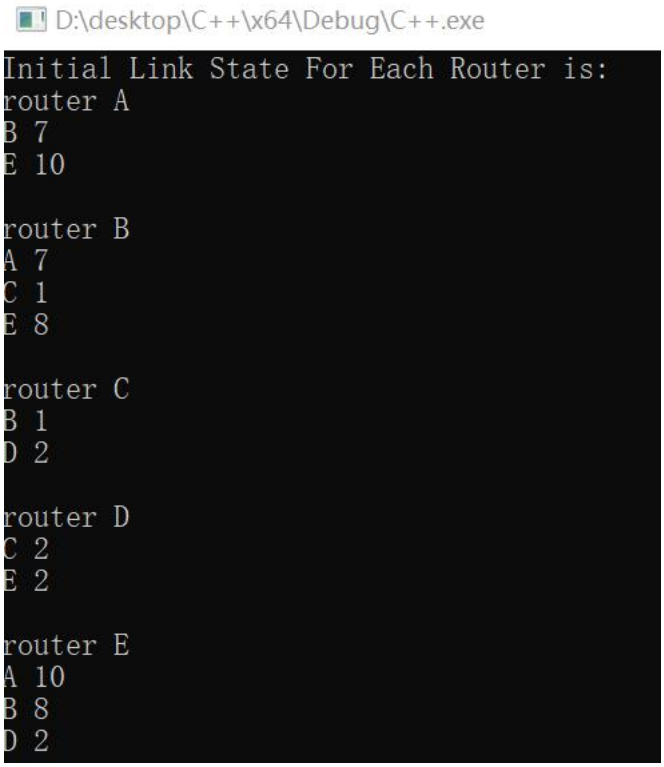
以 C++代码为例，Java 和 Python 的输出完全一致。

网络编程实验内容中的邻接矩阵

0 7 99 99 10
7 0 1 99 8
99 1 0 2 99
99 99 2 0 2
10 8 99 2 0 对应的网络结构为：



1. 运行程序，输出初始网络拓扑（配置文件中）对应的各个路由器的链路状态



2. 根据最短路径优先 SPF 算法计算任意路由器之间的最短路径，输出每个路由器每步的最短通路结果

```
Shortest Way Of Each Step For Each Router is:
For Router A
Step 0: A-B 7
Step 1: A-B-C 8
Step 2: A-B-C-D 10
Step 3: A-E 10

For Router B
Step 0: B-C 1
Step 1: B-C-D 3
Step 2: B-C-D-E 5
Step 3: B-A 7

For Router C
Step 0: C-B 1
Step 1: C-D 2
Step 2: C-D-E 4
Step 3: C-B-A 8

For Router D
Step 0: D-C 2
Step 1: D-E 2
Step 2: D-C-B 3
Step 3: D-C-B-A 10

For Router E
Step 0: E-D 2
Step 1: E-D-C 4
Step 2: E-D-C-B 5
Step 3: E-A 10
```

3. 输出最终态各个路由器的路由表

```
Final Routing Table For Each Router is:
Router A
A 0
B 7
B 8
B 10
E 10

Router B
A 7
B 0
C 1
C 3
C 5

Router C
B 8
B 1
C 0
D 2
D 4

Router D
C 10
C 3
C 2
D 0
E 2

Router E
A 10
D 5
D 4
D 2
E 0
```

7. 实验总结

这个实验并不难，dijkstra 算法之前接触过很多次，但是这道题中要求输出每个路由器每步的最短通路结果，即输出每个路由器到其他路由器的最短路径，和之前有点不一样。我采用的方法是用一个二维的 nextRouter 数组保存从 i 到 j 的下一跳，对每个路由器调用一次 dijkstra 算法的时候，从 i 到 j 的路径中每一跳都保留到目标 j 的下一跳，之后再 while 循环打印路径。