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 1　距离矢量（DV）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距离矢量路由算法，加深对 DV 算法的理解和掌握。

2. 实验内容

编写一个基于 DV 的路由算法，定期更新路由表。

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 16，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路由表，之后更新路由表，再输出最终态各个路由器的路由表。根据用户输入，更新网络拓扑或者输出对应路由器的路由表或者更新路由表。要反应出好消息传播的快，坏消息传播的慢这个特征。

3. 实验原理

在距离矢量路由算法中，所有节点都定期地将它们的整个路由选择表传送给所有与之直接相邻的节点，这种路由选择表包含：

- 每条路径的下一节点
- 路径的距离

在这种算法中，所有节点都必须参与距离矢量交换，以保证路由的有效性和一致性，也就是说，所有的节点都监听从其他节点传来的路由选择更新信息，并在下列情况下更新它们的路由选择表。

- 被通告一条新的路由，该路由在本节点的路由表中不存在，此时本地系统加入这条新的路由。
- 发来的路由消息中有一条到达某个目的地的路由，该路由与当前使用的路由相比，有较短的距离。这种情况下，就用经过发送路由信息的节点的新路由替换路由表中到达目的地的所有路由。

距离矢量路由算法的实质是，迭代计算一条路由中的站段数或延迟时间，从而得到到达一个目标的最短通路。它要求每个节点在每次更新时都将它的全部路由表发送给所有相邻的节点。显然，更新报文的大小与通信子网的节点个数成正比，大的通信子网将导致很大的更新报文。由于更新报文发送给直接相邻的借点，所以所有节点都将参加路由选择信息交换。基于这些原因，在通信子网上传送的路由选择信息的数量很容易变得非常大。

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，以C++代码为例进行分析。

代码实现的功能是：将编程实验安排中的邻接矩阵初始化作为配置文件，然后输出该邻接矩阵对应的各个路由器的路由表，之后更新路由表，输出最终态各个路由器的路由表。之后根据用户输入（用户可以无限输入直至退出），每次输入三个数（num1 num2 distance）表示将序号为 num1 和 num2 节点间的距离更新为 distance，然后更新网络拓扑结构，输出网络稳定后各个路由器的路由表。

• 全局变量

```
private:
    char nextRouter[n][n];
    string s = "ABCDEFGHIJKLMN";
public:
    int adjMat[n][n] = {
        { 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 oldTableMat[n][n];
```

• 打印每个路由器的路由表

```
void printRoutingTable()
    {
        for (int i = 0; i < n; i++)
        {
            cout << "Router " << s[i] << endl;
            for (int j = 0; j < n; j++)
            {
                cout << nextRouter[i][j] << " " << oldTableMat[i][j] << endl;
            }
            cout << endl;
        }
    }
```

• 打印初始邻接矩阵对应网络的路由表

```
void printInitialRoutingTable()
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
```

```
            {
                oldTableMat[i][j] = adjMat[i][j];
                if (oldTableMat[i][j] != 99)
                {
                    nextRouter[i][j] = s[j];
                }
                else
                {
                    nextRouter[i][j] = ' ';
                }
            }
        }
        cout << "Initial Routing Table For Each Router is: " << endl;
        printRoutingTable();
    }
```

• 新旧邻接矩阵进行比较，如果不相同，就更新，否则表示已收敛，退出更新

```
bool compare(int newTableMat[n][n])
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (oldTableMat[i][j] != newTableMat[i][j])
                {
                    return false;
                }
            }
        }
        return true;
    }
```

• 传递信息到相邻节点，循环更新，直至收敛，打印更新次数和更新后的路由表

```
void printAfterUpdateRoutingTable()
    {
        int time = 0;
        while (true)
        {
            time++;
            int newTableMat[n][n];
            memset(newTableMat, 0, sizeof(newTableMat));
            for (int i = 0; i < n; i++)//源节点为 i
            {
                for (int j = 0; j < n; j++)//目标节点为 j
                {
                    if (i == j)
```

```
                {
                    nextRouter[i][j] = s[i];
                    continue;
                }
                int min = 99;
                char c = ' ';
                for (int k = 0; k < n; k++)//如果 i 到 j 的下一跳为 k
                {
                    if (adjMat[i][k] != 0 && adjMat[i][k] != 99)
                    {
                        if (oldTableMat[k][j] != 99)
                        {
                            int temp = adjMat[i][k] + oldTableMat[k][j];
                            if (temp < min)
                            {
                                min = temp;
                                c = s[k];
                            }
                        }
                    }
                }
                //更新新路由表和下一跳
                newTableMat[i][j] = min;
                nextRouter[i][j] = c;
            }
        }
        if (compare(newTableMat) == false)//新旧路由表进行比较
        {
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    oldTableMat[i][j] = newTableMat[i][j];
                }
            }
        }
        else
        {
            printf("Convergence costs %d times of exchanging information.\n", time);
            cout << "After-update Routing Table For Each Router is: " << endl;
            printRoutingTable();
            break;
        }
}}
```

·主函数

```
int main()
{
    DV operation;
    operation.printInitialRoutingTable();
    operation.printAfterUpdateRoutingTable();
    while (true)
    {
        cout << "Enter 0 to exit or enter 1 to upgrade topology network:" << endl;
        int a;
        cin >> a;
        if (a == 0)
        {
            break;
        }
        else if (a == 1)
        {
            cout << "Please input as format 'num1 num2 distance' such as '0 1 8'(no
'') to upgrade a pair of nodes:" << endl;
            int num1, num2, distance;
            cin >> num1 >> num2 >> distance;
            operation.adjMat[num1][num2] = distance;
            operation.adjMat[num2][num1] = distance;
            operation.oldTableMat[num1][num2] = distance;
            operation.oldTableMat[num2][num1] = distance;
            cout << endl;
            operation.printAfterUpdateRoutingTable();
        }

    }
    system("pause");
}
```

6. 实验结果

以 C++代码为例，Java 和 Python 的输出完全一致。

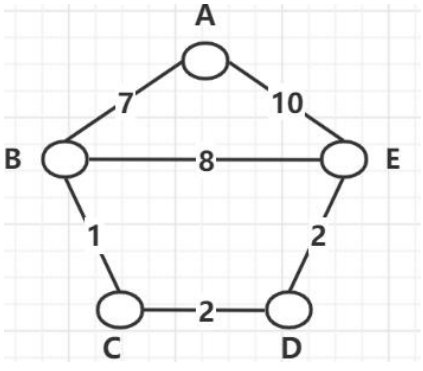
0 7 99 99 10

7 0 1 99 8

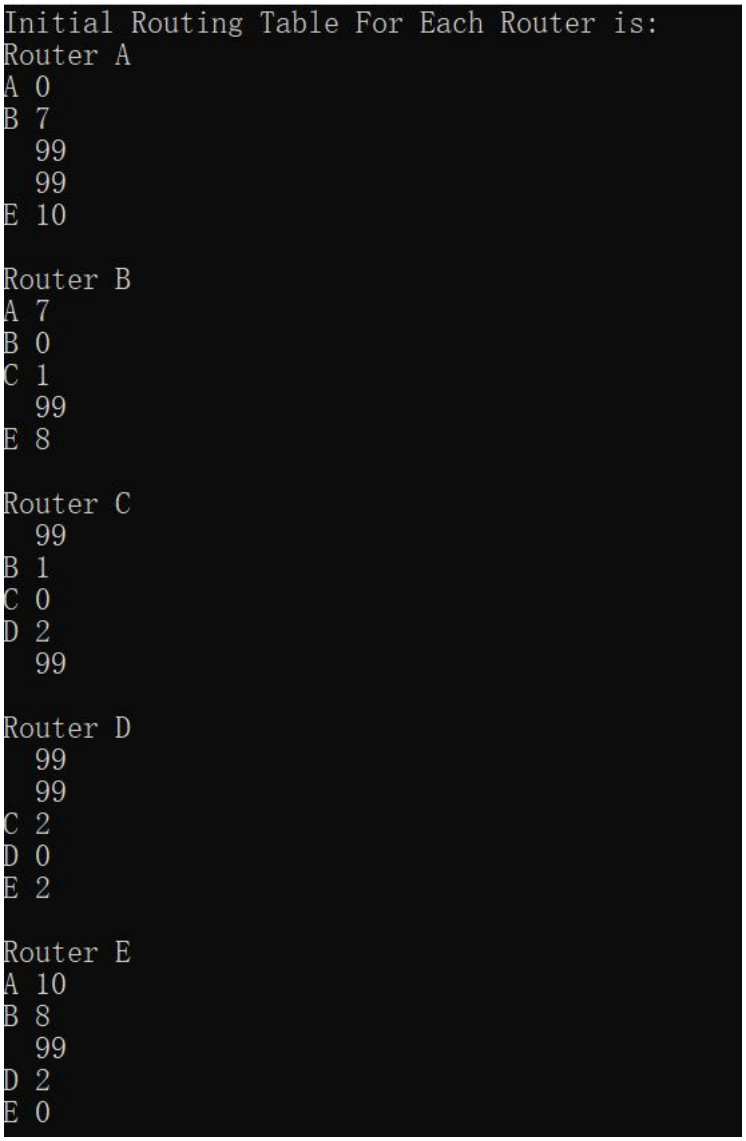
99 1 0 2 99

99 99 2 0 2

10 8 99 2 0 对应的网络结构为：



邻接矩阵对应的初始输出为：



此时尚未进行任何信息交换，每个路由器只知道与它相邻路由器之间的距离。

更新后每个路由器对应的路由表为：

A		B		C		D		E	
A	0	A	7	B	8	C	10	A	10
B	7	B	0	B	1	C	3	D	5
B	8	C	1	C	0	C	2	D	4
B	10	C	3	D	2	D	0	D	2
E	10	C	5	D	4	E	2	E	0

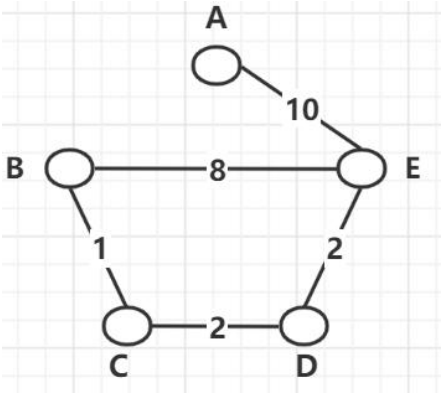
每个路由器的路由表的第一列为下一跳，第二列为到顺序目的地的距离。

输出为：

```
Convergence costs 3 times of exchanging information.
After-update 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
```

收敛次数为 3 次，完成信息交换

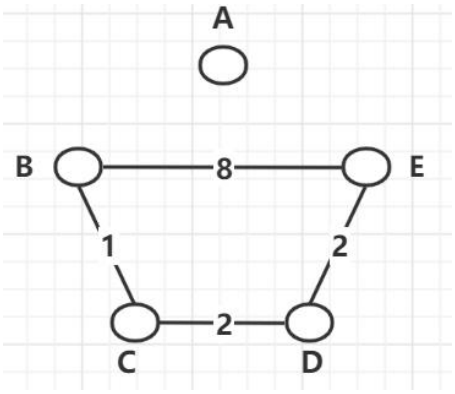
1. 第一次更新，断开 A 和 B，输入 0 1 99




```
Enter 0 to exit or enter 1 to upgrade topology network:
1
Please input as format 'num1 num2 distance' such as '0 1 8'
0 1 99

Convergence costs 7 times of exchanging information.
After-update Routing Table For Each Router is:
Router A
A 0
E 15
E 14
E 12
E 10

Router B
C 15
B 0
C 1
C 3
C 5

Router C
D 14
B 1
C 0
D 2
D 4

Router D
E 12
C 3
C 2
D 0
E 2

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

收敛次数为 7 次

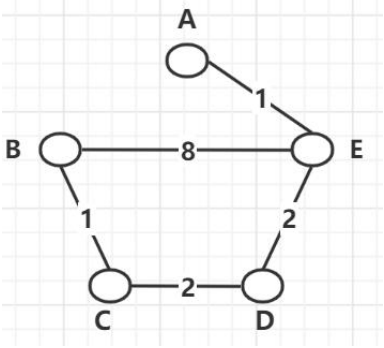
2. 第二次更新，断开 A E，此时 A 成为孤立点，输入 0 4 99




```
Enter 0 to exit or enter 1 to upgrade topology network:
1
Please input as format 'num1 num2 distance' such as '0 1
0 4 99

Convergence costs 87 times of exchanging information.
After-update Routing Table For Each Router is:
Router A
A 0
  99
  99
  99
  99

Router B
  99
B 0
C 1
C 3
C 5

Router C
  99
B 1
C 0
D 2
D 4

Router D
  99
C 3
C 2
D 0
E 2

Router E
  99
D 5
D 4
D 2
E 0
```

收敛次数为 87 次，可见坏消息传播慢这个特征

3. 第三次更新，连接 A E，输入 0 4 1




```
Enter 0 to exit or enter 1 to upgrade topology network:
1
Please input as format 'num1 num2 distance' such as '0 1
0 4 1

Convergence costs 4 times of exchanging information.
After-update Routing Table For Each Router is:
Router A
A 0
E 6
E 5
E 3
E 1

Router B
C 6
B 0
C 1
C 3
C 5

Router C
D 5
B 1
C 0
D 2
D 4

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

Router E
A 1
D 5
D 4
D 2
E 0
```

收敛次数为 4 次，可见好消息传播快这个特征

7. 实验总结

这个实验原理简单，但实现起来并不好弄，调试了许久。每次由邻接矩阵和旧的路由表计算出新的路由表，如果旧路由表和新路由表完全一致，表示已收敛，退出循环；否则将新路由表作为旧路由表，新路由表置零，继续循环。这是个 n**3 的计算过程，可见比较耗时。在上述实验中可以看出，断开得到孤立点的收敛次数为 87 次，连接孤立点的收敛次数为 4 次，反应出好消息传播的快，坏消息传播的慢这个特征。