



Distance-Vector Algorithm

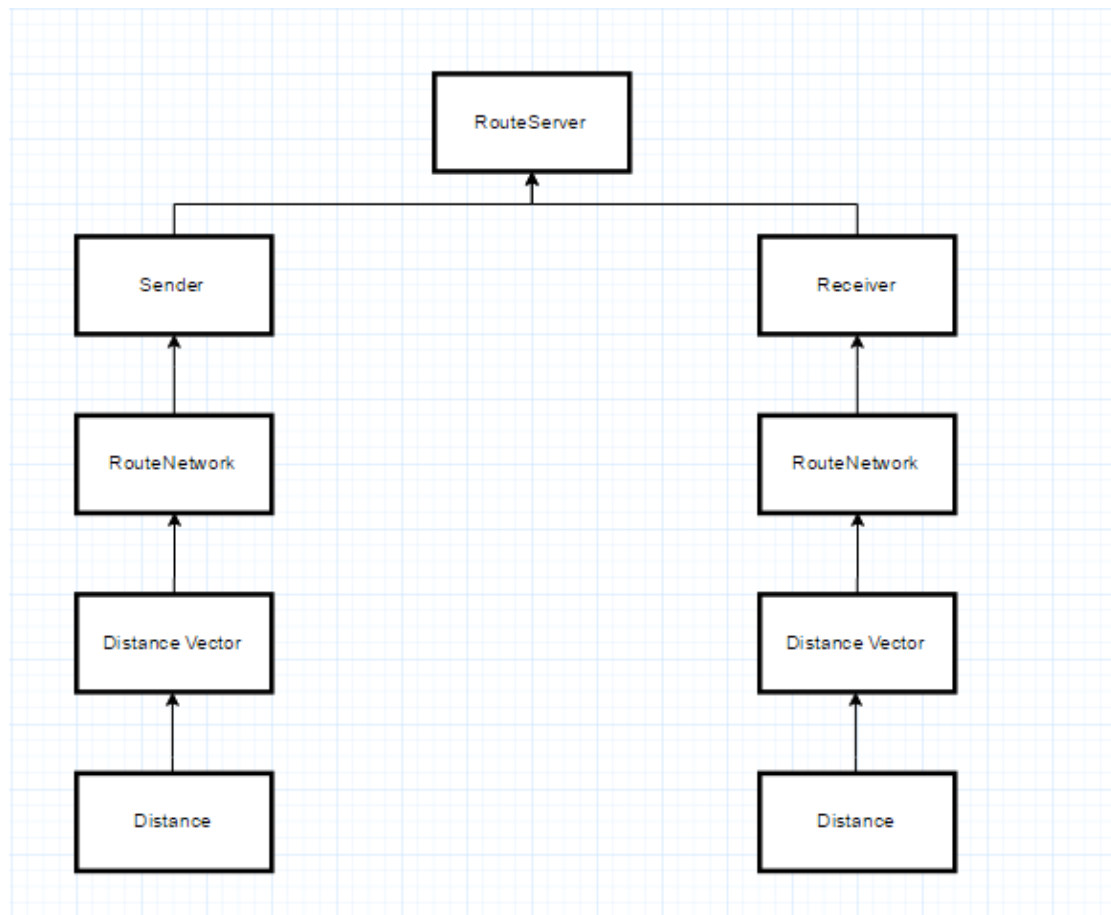
指導教授：張英超 教授

學生：資管四 S0261001 周平

資管四 S0261020 蘇揮原



一、 本次作業的架構圖及配分



《程式架構圖》

組員配分：資管四 S0261020 周平(50%)、資管四 S0261020 蘇揮原(50%)

延伸功能：每個節點會請求作業系統自動配置 port number

```
1 import java.io.BufferedReader;
2
3 /*****
4  *
5  *      主程式輸入點，可以指定任何階級，但是總台路由階級要相同
6  *
7  *****/
8
9 // S0261001 資管四 周平(50%)、S0261020 資管四 蔡耀原(50%)
10 // 延伸功能：每個節點會自動至 port number 無需自行輸入
11
12
13 class S0261001
14 {
15     public static void main(String args[]) throws Exception
16     {
17         char local;
18         String file_name;
19
20         /* 輸入節點名稱及輸入檔案名稱 */
21         Scanner sc = new Scanner(System.in);
22         System.out.print("[設定訊息]:: 本節點的名稱: ");
23         local = sc.next().charAt(0);
24         System.out.print("[設定訊息]:: 輸入檔的名稱: ");
25         file_name = sc.next();
26
27         RouteNetwork network = new RouteNetwork(local, file_name);
28         ReentrantLock lock = new ReentrantLock(); // lock 用來控制 concurrency
29         DatagramSocket socket = new DatagramSocket();
30
31         System.out.println("[設定訊息]:: 請選擇以下選項: ");
32         System.out.println("[設定訊息]:: [1]: Network Initialization ");
33         System.out.println("[設定訊息]:: [2]: Link cost update ");
34         System.out.println("[設定訊息]:: [3]: Exit the program ");
35         System.out.print("[設定訊息]:: [_]: 你的選擇: ");
36         network.mode = sc.nextInt();
37
38         /* 判斷目前要執行的設定階段 */
39         while(network.mode != 0)
40         {
41             if(network.mode == 1)
42             {
43
44             }
```

```
46     BufferedReader uip = new BufferedReader(new InputStreamReader(System.in));
47
48     System.out.println("[通知訊息]: 提示: 請在 30s內輸入完所有 IP 和 Port 否則接收端會視為穩定結束");
49     System.out.println("[通知訊息]: 提示: 最後訊息傳送後30秒才會達到穩定狀態, 凡請耐心等待, 切勿關閉");
50     System.out.println("[通知訊息]: 提示: 以下輸入的 IP 和 Port 要依照讀檔順序輸入");
51     System.out.println("[Receiver 訊息]: Socket 狀態: socket 已經準備完成");
52     System.out.println("[Receiver 訊息]: Local 位址: " + InetAddress.getLocalHost());
53     System.out.println("[Receiver 訊息]: Local 埠號: " + socket.getLocalPort());
54     network.init = 1; // init 代表 Receiver 端的初始化是否完成
55     System.out.println(network.vect);
56
57     for(int i = 0; i < network.node - 1; i++) // 輸入連續的節點資訊
58     {
59         System.out.print("[Sender 訊息]: 請輸入第 " + (i + 1) + " 個節點的 IP addr : ");
60         String ServerIP = uip.readLine();
61         System.out.print("[Sender 訊息]: 請輸入第 " + (i + 1) + " 個節點的 IP port : ");
62         network.port_table[i] = new Integer(uip.readLine());
63         network.addr_table[i] = InetAddress.getByName(ServerIP);
64     }
65     network.connect = 1;
66
67     Sender sdr = new Sender(network, lock);
68     Receiver rcv = new Receiver(network, lock, socket);
69     rcv.start();
70     sdr.start();
71     rcv.join();
72     sdr.join();
73 }
74 else if(network.mode == 2)
75 {
76     System.out.print("[設定訊息]: 新的輸入檔的名稱: ");
77     file_name = sc.next();
78     network.updateVectCost(file_name);
79
80     network.over = 0;
81
82     Sender sdr = new Sender(network, lock);
83     Receiver rcv = new Receiver(network, lock, socket);
84     rcv.start();
85     sdr.start();
86     rcv.join();
87     sdr.join();
88 }
```

```
85         sdr.start();
86         rcv.join();
87         sdr.join();
88     }
89     else if(network.mode == 3)
90     {
91         System.out.print("[通知聲響]: 已經結束程式 ");
92         Sender sdr = new Sender(network, lock);
93         Receiver rcv = new Receiver(network, lock, socket);
94         rcv.start();
95         sdr.start();
96         rcv.join();
97         sdr.join();
98
99         Thread.sleep(1000);
100        break;
101    }
102
103    System.out.println("[設定訊息]: 請選擇以下選項: ");
104    System.out.println("[設定訊息]: [1]: Network Initialization ");
105    System.out.println("[設定訊息]: [2]: Link cost update ");
106    System.out.println("[設定訊息]: [3]: Exit the program ");
107    System.out.print("[設定訊息]: [_]: 你的選擇: ");
108    network.mode = sc.nextInt();
109 }
110 sc.close();
111 }
112 }
```

Class RouteNetwork: 記錄使用到的所有網路資訊及距離資訊

```
1 import java.io.*;
2 import java.net.*;
3 import java.util.*;
4
5 /**
6  *
7  *      紀錄網路的資訊及透過發送端及接收端
8  *
9  */
10
11 class RouteNetwork
12 {
13     public int mode;           // 紀錄網路目前的狀態
14     public int init;           // 紀錄網路初始化狀態
15     public int over;           // 紀錄網路結束狀態
16     public int node;           // 紀錄節點數量資訊
17     public int link;           // 紀錄鏈路數量資訊
18     public int connect;        // 紀錄連線初始化狀態
19     public char name_table[];   // 紀錄節點名稱資訊
20     public InetAddress addr_table[]; // 紀錄 IP 位址資訊
21     public int port_table[];    // 紀錄埠號資訊
22     public DistanceVector vect; // 紀錄距離矩陣資訊
23     public String msgBuffer[];  // 紀錄待傳送的消息
24     public int bufferPointer;    // 待傳送消息的指標
25
26     public RouteNetwork(char local, String file_name) throws Exception
27     {
28         mode = 0;
29         init = 0;
30         node = 0;
31         link = 0;
32         connect = 0;
33         msgBuffer = new String[20];
34         bufferPointer = -1;
35
36         Scanner file = new Scanner(new FileReader(file_name));
37         String cntseq = file.nextLine();
38         for(int i = 0; i < cntseq.length(); i++)
39         {
40             /* 讀入的內容若包含英文字元，則表示是節點名稱，否則為成本 */
41             if(cntseq.charAt(i) >= 'a' && cntseq.charAt(i) <= 'z')
42             {
43                 node++;
44             }
45         }
46     }
47 }
```

```

44     }
45 }
46 while(file.hasNext())
47 {
48     file.nextLine();
49     link++;
50 }
51
52 System.out.println("[通知訊息]: 找到節點數量: " + node + ", 鏈結數量: " + link);
53
54 this.name_table = new char[node];
55 this.addr_table = new InetAddress[node];
56 this.port_table = new int[node];
57
58 file = new Scanner(new FileReader(file_name));
59 String sequence = file.nextLine();
60
61 /* 成本和成本之間以空白做區隔, 因此每次位移2個字元 */
62 for(int i = 0; i < node; i++)
63 {
64     name_table[i] = sequence.charAt(i * 2);
65 }
66
67 this.vect = new DistanceVector(node, local, name_table);
68 this.vect.dist_table = new Distance[link]; // 初始化成本陣列
69
70 /* 把讀入的節點和成本資訊儲存 */
71 for(int i = 0; i < link; i++)
72 {
73     char node1 = file.next().charAt(0);
74     char node2 = file.next().charAt(0);
75     int distance = file.nextInt();
76     vect.dist_table[i] = new Distance(node1, node2, distance);
77 }
78 vect.init();
79
80 /* 輸出初始化的距離矩陣 */
81 System.out.println("[通知訊息]: 建立初始狀態表格如下所示:");
82 printVector();
83
84 file.close();
85 }
86 public void updateVectCost(String file_name) throws Exception
87 {

```

```
86 public void updateVectCost(String file_name) throws Exception
87 {
88     this.vect.updateCost(file_name);
89 }
90 public void printVector()
91 {
92     System.out.println(this.vect);
93 }
94 }
```

Class RouteNetwork 程式碼(3/3)

Class Receiver: 負責接收封包及設定確認封包工作

```
1 import java.net.*;
2
3 /*****
4  *
5  *      處理路由器的接收檔案動作及製作待回傳的封包內容
6  *
7  *****/
8
9 import java.util.concurrent.locks.ReentrantLock;
10 public class Receiver extends Thread
11 {
12     RouteNetwork network;
13     ReentrantLock lock;
14     DatagramSocket socket;
15     public void run()
16     {
17         if(network.mode == 1) // 網路狀態 1. 代表目前為初始化階段
18         {
19             try{
20                 socket.setSoTimeout(30000); // 設定 socket timeout 時間為 60 秒
21                 for(;;)
22                 {
23                     String tmp;
24                     byte buffer[] = new byte[200];
25                     DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
26                     socket.receive(packet);
27                     tmp = new String(buffer, 0, packet.getLength());
28                     String msg = tmp.split("_")[1]; // 以 '_' 作為標記符號代表欄位
29                     lock.lock();
30                     System.out.println("[Receiver 訊息]: 收到距離更新: " + msg);
31                     String vect[] = msg.split(" ");
32                     int value[] = new int[vect.length - 1];
33                     for(int i = 1; i < vect.length; i++) // 處理送來的距離資訊
34                     {
35                         value[i - 1] = new Integer(vect[i]);
36                     }
37
38                     network.vect.setRow(msg.charAt(0), value); // 更新距離資訊
39                     network.vect.update(); // 計算新的距離矩陣
40                     System.out.println(network.vect);
41                     lock.unlock();
```

```

42     }
43     }catch (SocketTimeoutException e) {
44         System.out.println(network.vect.routingTable());
45         System.out.println(network.vect.local + " 的 Distance Vector 演算法已經穩定");
46         network.over = 1;
47     }catch (Exception e){
48         System.out.println(e);
49     }
50 }
51 else if(network.mode == 2) // 網路狀態 2. 代表目前為重新輸入權階級
52 {
53     try{
54         socket.setSoTimeout(20000);
55         for(;;)
56         {
57             String rawMsg;
58             String processedMsg[];
59             byte buffer[] = new byte[400];
60             DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
61             socket.receive(packet);
62             rawMsg = new String(buffer, 0, packet.getLength());
63             processedMsg = rawMsg.split("_"); // 以 '_' 符號作為欄位區隔
64             Thread.sleep(500); // 延遲 0.5s 先讓 Sender 拿到 lock 重新資料, 避免兩次更新合併為一次
65             lock.lock();
66             if(processedMsg[0].equals("1")) // 若封包內容開頭為 1 則為延遲重新資訊
67             {
68                 for(int i = 1; i < processedMsg.length; i++)
69                 {
70                     System.out.println("[Receiver 訊息]: 收到延遲更新:" + processedMsg[i]);
71                 }
72                 String vect[] = processedMsg[processedMsg.length - 1].split(" ");
73                 int value[] = new int[vect.length - 1];
74                 for(int i = 1; i < vect.length; i++) // 處理送來的連續資訊
75                 {
76                     value[i - 1] = new Integer(vect[i]);
77                 }
78
79                 network.vect.setRow(processedMsg[processedMsg.length - 1].charAt(0), value);
80                 network.vect.update();
81                 System.out.println(network.vect.toString());
82             }

```

```

82
83         if(processedMsg.length >= 3) // 製作確認收到訊息的資訊放到buffer 等待發送
84         {
85             network.bufferPointer++;
86             network.msgBuffer[network.bufferPointer] = "2_" + processedMsg[processedMsg.length - 1].charAt(0) + "_";
87             for(int i = 1; i < processedMsg.length - 1; i++)
88             {
89                 network.msgBuffer[network.bufferPointer] = network.msgBuffer[network.bufferPointer] + "[Receiver 訊息]: 節點" + network.vec
90             }
91         }
92     }
93     if(processedMsg[0].equals("2")) // 當封包開頭為2代表為確認收到訊息資訊
94     {
95         for(int i = 2; i < processedMsg.length; i++)
96         {
97             System.out.println(processedMsg[i]);
98         }
99     }
100    if(processedMsg[0].equals("3")) // 當封包開頭為3代表為節點中斷連續資訊
101    {
102        System.out.println(processedMsg[1]);
103    }
104    lock.unlock();
105 }
106 }catch (SocketTimeoutException e) {
107     System.out.println(network.vect.routingTable());
108     System.out.println(network.vect.local + " 的 Distance Vector 演算法已經穩定");
109     network.over = 1;
110 }catch (Exception e){
111     System.out.println("Receiver 發生錯誤");
112     System.out.println(e);
113 }
114 }
115 }
116 public Receiver(RouteNetwork network, ReentrantLock lock, DatagramSocket socket)
117 {
118     this.lock = lock;
119     this.network = network;
120     this.socket = socket;
121 }
122 }

```

Class Sender: 負責發送封包

```
1 import java.io.*;
2
3 /*****
4  *
5  *      處理路由器的發送動作及回應確認工作
6  *
7  *****/
8
9 import java.net.*;
10 import java.util.concurrent.locks.ReentrantLock;
11 public class Sender extends Thread
12 {
13     RouteNetwork network;
14     ReentrantLock lock;
15     public void run()
16     {
17         if(network.mode == 1) // network.mode 1代表Sender 初始化階段
18         {
19             try{
20                 while (true) {
21                     lock.lock();
22                     if(network.vect.dirty == true) // 檢查資料有無更新，有更新才需要發送
23                     {
24                         String msg = network.vect.getMsg();
25                         int oLength = msg.getBytes().length;
26                         byte buffer[] = new byte[oLength];
27                         buffer = msg.getBytes();
28                         for(int i = 0; i < network.node - 1; i++)
29                         {
30                             int portNo = network.port_table[i];
31                             InetAddress addr = network.addr_table[i];
32                             DatagramPacket packet = new DatagramPacket(buffer, oLength, addr, portNo);
33                             DatagramSocket socket = new DatagramSocket();
34                             socket.send(packet);
35                             socket.close();
36                         }
37                         network.vect.dirty = false;
38                     }
39                     lock.unlock();
40                     if(network.over == 1) // network.over 代表是否已經 timeout(穩定) 若穩定則結束 Sender
41                     {
```

```

42         break;
43     }
44 }
45 }catch (Exception e){
46     System.out.println("-網路發生錯誤-");
47 }
48 }
49 else if(network.mode == 2) // network.mode = 2代表 Sender 重新輸入權階級
50 {
51     try{
52         while (true) {
53             lock.lock();
54             if(network.vect.dirty == true) // 檢查自己的狀態是否有更新，若有更新就要發送
55             {
56                 String msg = network.vect.updateMsg;
57                 int oLength = msg.getBytes().length;
58                 byte buffer[] = new byte[oLength];
59                 buffer = msg.getBytes();
60                 for(int i = 0; i < network.node - 1; i++)
61                 {
62                     int portNo = network.port_table[i];
63                     InetAddress addr = network.addr_table[i];
64                     DatagramPacket packet = new DatagramPacket(buffer, oLength, addr, portNo);
65                     DatagramSocket socket = new DatagramSocket();
66                     socket.send(packet);
67                     socket.close();
68                 }
69                 network.vect.dirty = false;
70             }
71             lock.unlock();
72             break;
73         }
74     }
75     while (true) {
76         lock.lock();
77         if(network.vect.dirty == true) // 檢查距離短端是否有更新，若有更新就要發送新的資訊
78         {
79             String msg = network.vect.getMsg();
80             int oLength = msg.getBytes().length;
81             byte buffer[] = new byte[oLength];
82             buffer = msg.getBytes();

```

```

81 byte buffer[] = new byte[oLength];
82 buffer = msg.getBytes();
83 for(int i = 0; i < network.node - 1; i++)
84 {
85     int portNo = network.port_table[i];
86     InetAddress addr = network.addr_table[i];
87     DatagramPacket packet = new DatagramPacket(buffer, oLength, addr, portNo);
88     DatagramSocket socket = new DatagramSocket();
89     socket.send(packet);
90     socket.close();
91 }
92 network.vect.dirty = false;
93 }
94 while(network.bufferPointer >= 0) // 檢查是否有確認訊息尚未傳送，若有就先傳送
95 {
96     String proccessedMsg[] = network.msgBuffer[network.bufferPointer].split("_");
97
98     int oLength = network.msgBuffer[network.bufferPointer].getBytes().length;
99     byte buffer[] = new byte[oLength];
100     buffer = network.msgBuffer[network.bufferPointer].getBytes();
101     int destId = network.vect.getIndex(proccessedMsg[1].charAt(0));
102
103     /* 處理待傳送的確認訊息要傳送的對象 */
104     if(network.vect.getIndex(proccessedMsg[1].charAt(0)) > network.vect.getIndex(network.vect.local))
105     {
106         destId = destId - 1; // destId 就是要傳送對象的索引index
107     }
108
109     int portNo = network.port_table[destId];
110     InetAddress addr = network.addr_table[destId];
111     DatagramPacket packet = new DatagramPacket(buffer, oLength, addr, portNo);
112     DatagramSocket socket = new DatagramSocket();
113     socket.send(packet);
114     socket.close();
115     network.bufferPointer--;
116 }
117 lock.unlock();
118 if(network.over == 1)
119 {
120     break;
121 }

```

```

122     }
123     }catch(Exception e){
124         System.out.println("發送端發生錯誤");
125         System.out.println(e);
126     }
127 }
128 else if(network.mode == 3) // network.mode = 3 代表準備結束，要發送中斷連續資訊給其他節點
129 {
130     try{
131         while (true) {
132             lock.lock();
133             String msg = "3_[通知訊息]:: 節點" + network.vect.local + " 已經結束運作";
134             int oLength = msg.getBytes().length;
135             byte buffer[] = new byte[oLength];
136             buffer = msg.getBytes();
137             for(int i = 0; i < network.node - 1; i++)
138             {
139                 int portNo = network.port_table[i];
140                 InetAddress addr = network.addr_table[i];
141                 DatagramPacket packet = new DatagramPacket(buffer, oLength, addr, portNo);
142                 DatagramSocket socket = new DatagramSocket();
143                 socket.send(packet);
144                 socket.close();
145             }
146             network.vect.dirty = false;
147             lock.unlock();
148             break;
149         }
150     }catch(Exception e){
151         System.out.println("發送端發生錯誤");
152         System.out.println(e);
153     }
154 }
155
156 public Sender(RouteNetwork network, ReentrantLock lock)
157 {
158     this.lock = lock;
159     this.network = network;
160 }
161 }
162

```

```

1 import java.io.*;
2
3 /**
4  *
5  *      DistanceVector 用來記錄距離資訊及相關資料的輸入、輸出
6  *
7  */
8
9 import java.util.*;
10 class DistanceVector
11 {
12     public int size;                // 記錄總結點數量
13     public int vect[][];            // 記錄節點到節點的距離矩陣
14     public char local;              // 記錄本節點的名稱
15     public char next_hop[];         // 記錄next hop 節點
16     public char sequence[];         // 記錄所有節點的名稱
17     public boolean dirty;           // 記錄矩陣是否被修改
18     public String updateMsg;        // 記錄通知更新訊息
19     public Distance dist_table[];   // 記錄所有link 當前的資訊
20
21     public void init() // 初始化距離矩陣的值
22     {
23         for(int i = 0; i < size; i++)
24         {
25             for(int j = 0; j < size; j++)
26             {
27                 if(sequence[i] == local)
28                 {
29                     vect[i][j] = getCost(local, sequence[j]);
30                 }
31                 else
32                 {
33                     vect[i][j] = -1;
34                 }
35             }
36         }
37     }
38     public DistanceVector(int size, char local, char seq[]) // 預設建構子
39     {
40         this.size = size;
41         this.vect = new int[size][size];
42         this.local = local;
43         this.next_hop = new char[size];

```

Class DistanceVector: 負責儲存距離矩陣以及提供設定、存取、輸出等涵式方便更新及輸出表格使用

```

44     this.sequence = new char[size];
45     this.dirty = true;
46
47     /* 初始狀態將每個節點的 next hop 設為其本身 */
48     for(int i = 0; i < seq.length; i++)
49     {
50         sequence[i] = seq[i];
51         next_hop[i] = seq[i];
52     }
53 }
54 public int getIndex(char name) // 取得該節點名稱對應的 index
55 {
56     /* 搜尋名稱所在的索引，若搜尋不到則回傳 -1 */
57     for(int i = 0; i < size; i++)
58     {
59         if(sequence[i] == name)
60         {
61             return i;
62         }
63     }
64     return -1;
65 }
66 public int[] getRow(char node) // 取得該節點的距離資訊
67 {
68     int idx = getIndex(node);
69     int dist[] = new int[4];
70     for(int i = 0; i < dist.length; i++)
71     {
72         dist[i] = vect[idx][i];
73     }
74     return dist;
75 }
76 public void setRow(char node, int dist[]) // 設定該節點的距離資訊
77 {
78     int idx = getIndex(node);
79     for(int i = 0; i < dist.length; i++)
80     {
81         boolean modified = false;
82         /* 若連結距離為 -1 代表無限大，任何非 -1 的連結都可以取代 */
83         if(vect[idx][i] != -1 && vect[idx][i] < dist[i])
84         {
85             vect[getIndex(local)][i] = getCost(local, sequence[i]);
86             next_hop[i] = sequence[i];
87             modified = true;

```

```

86         next_hop[i] = sequence[i];
87         modified = true;
88     }
89     vect[idx][i] = dist[i];
90     if(modified == true)
91     {
92         update();
93         dirty = true;
94     }
95 }
96 }
97 public void update() // 重新計算新的距離矩陣
98 {
99     int idx = getIndex(local);
100     for(int i = 0; i < size; i++) // 對每一個目的地做以下動作
101     {
102         if(i != idx) // 若該節點不是本機本身則做以下動作
103         {
104             int min = vect[idx][i];
105
106             for(int j = 0; j < size; j++) // 以每一個節點作為 next hop
107             {
108                 if(j != idx && j != i) // 若 hop 不是本機本身
109                 {
110                     int hop = getIndex(sequence[j]);
111                     int cost = getCost(local, sequence[j]);
112
113                     if(getCost(local, sequence[j]) >= 0 && vect[hop][i] >= 0)
114                     if(cost + vect[hop][i] < min || min == -1) // 若新的距離小於原本的距離則將其記錄
115                     {
116                         min = cost + vect[hop][i];
117                         next_hop[i] = sequence[j];
118                         dirty = true;
119                     }
120                 }
121             }
122             if((min < vect[idx][i]) || vect[idx][i] == -1) // 若找到比目前距離小的距離則更新
123             {
124                 vect[idx][i] = min;
125                 dirty = true;
126             }
127             else
128             {
129                 if(getCost(local, sequence[i]) >= 0 && getCost(local, sequence[i]) < 0)

```



```

171         next_hop[getIndex(node1)] = node1;
172     }
173 }
174 else // 若其距離存在，暫時先將目前距離設為鏈結直接相關的距離，等待重新計算
175 {
176     if(node1 == local)
177     {
178         vect[getIndex(local)][getIndex(node2)] = distance;
179         next_hop[getIndex(node2)] = node2;
180     }
181     if(node2 == local)
182     {
183         vect[getIndex(local)][getIndex(node1)] = distance;
184         next_hop[getIndex(node1)] = node1;
185     }
186 }
187
188 dist_table[j] = new Distance(node1, node2, distance);
189 dirty = true;
190 break;
191 }
192 }
193 }
194 }
195
196 update(); // 計算完新的距離後需更新
197 int new_row[] = getRow(local);
198
199 for(int i = 0; i < size; i++) // 比對舊的距離和新的距離，若有改變則通知其他節點
200 {
201     if(old_row[i] != new_row[i])
202     {
203         if(old_row[i] != 9999)
204         {
205             System.out.println("[更新訊息]: 原本 " + local + " 到 " + sequence[i] + " 的成本由 " + old_row[i] + " 變成 " + new_row[i]);
206             updateMsg = updateMsg + "節點 " + local + " 訊息: 原本從 " + local + " 到 " + sequence[i] +
207                 " 的成本由 " + old_row[i] + " 變成 " + new_row[i] + "_";
208         }
209         else
210         {
211             System.out.println("[更新訊息]: 原本 " + local + " 到 " + sequence[i] + " 的成本由無限大變成 " + new_row[i]);
212             updateMsg = updateMsg + "節點 " + local + " 訊息: 原本從 " + local + " 到 " + sequence[i] +
213                 " 的成本由無限大變成 " + new_row[i] + "_";

```

```

214     }
215 }
216 }
217 updateMsg = updateMsg + getMsg().split("_")[1];
218 System.out.println(this.toString());
219 file.close();
220 }
221 public int getCost(char node1, char node2) // 取得兩節點直接連結的成本
222 {
223     if(node1 == node2)
224     {
225         return 0;
226     }
227     for(int i = 0; i < dist_table.length; i++)
228     {
229         if(dist_table[i].isPair(node1, node2))
230         {
231             return dist_table[i].distance;
232         }
233     }
234     return -1;
235 }
236 public String getMsg() // 取得距離矩陣的輸出資訊
237 {
238     String msg = "1_" + local + " ";
239     for(int i = 0; i < size; i++)
240     {
241         msg = msg + getRow(local)[i] + " ";
242     }
243     return msg;
244 }
245 public String routingTable() // 輸出路由表資訊
246 {
247     String msg = local + " 節點的路由表如下所示(目的/下個節點/成本):\n";
248     for(int i = 0; i < size; i++)
249     {
250         msg = msg + sequence[i] + " " + next_hop[i] + " " + vect[getIndex(local)][getIndex(sequence[i])] + "\n";
251     }
252     return msg;
253 }
254 public String toString() // output the content of DistanceVector
255 {
256     String output = " ";

```

```
256 String output = " ";
257 for(int i = 0; i < size; i++)
258 {
259     output = output + " " + sequence[i];
260 }
261 output = output + "\n";
262 for(int i = 0; i < size; i++)
263 {
264     output = output + sequence[i];
265     for(int j = 0; j < size; j++)
266     {
267         output = output + " " + vect[i][j];
268     }
269     output = output + "\n";
270 }
271 return output;
272 }
273 }
```

```

1  /*****
2  *
3  *      Distance 用來記錄 link 的資訊，包含兩端點及其距離
4  *
5  *****/
6
7  class Distance
8  {
9      public char node1; // 節點1
10     public char node2; // 節點2
11     public int distance;
12     public Distance(char node1, char node2, int distance) // 預設建構子
13     {
14         this.node1 = node1;
15         this.node2 = node2;
16         this.distance = distance;
17     }
18     public boolean isPair(char node1, char node2) // 判斷此 link 是否由 node1 及 node2 組成
19     {
20         if((this.node1 == node1 && this.node2 == node2) || (this.node1 == node2 && this.node2 == node1))
21             return true;
22         else
23             return false;
24     }
25     public String toString() // 輸出節點及距離資訊
26     {
27         return "It cost " + this.distance + " from " + node1 + " to " + node2;
28     }
29 }

```

Class Distance 紀錄最底層的成本資訊以及涵式
提供方便存取成本使用，以及輸出成本資訊

三、本次作業程式執行功能說明

S0261001 [Java Application] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (2017年5月7日 下午12:55:36)

```
[設定訊息]:: 本節點的名稱: u
[設定訊息]:: 輸入檔的名稱: data.txt
[通知訊息]:: 找到節點數量: 4, 總節數量: 6
[通知訊息]:: 建立初始狀態表格如下所示:

u v w x
u 0 2 5 1
v -1 -1 -1 -1
w -1 -1 -1 -1
x -1 -1 -1 -1

[設定訊息]:: 請選擇以下選項:
[設定訊息]:: [1]: Network Initialization
[設定訊息]:: [2]: Link cost update
[設定訊息]:: [3]: Exit the program
[設定訊息]:: [1]: 你的選擇: 1
[通知訊息]:: 提示: 請在 60s 內輸入所有 IP 和 Port 否則接收端會視為檔定結束
[Receiver 訊息]:: Socket 狀態: socket 已經準備完成
[Receiver 訊息]:: Local 位址: Jimmy/192.168.0.106
[Receiver 訊息]:: Local 埠號: 51103
```

(1) 輸入節點資訊

```
u v w x
u 0 2 5 1
v -1 -1 -1 -1
w -1 -1 -1 -1
x -1 -1 -1 -1
```

(3) 初始表格輸出

S0261001 [Java Application] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (2017年5月7日 下午12:55:36)

```
[Sender 訊息]:: 請輸入第 1 個節點的 IP addr : 192.168.0.106
[Sender 訊息]:: 請輸入第 1 個節點的 IP port : 51104
[Sender 訊息]:: 請輸入第 2 個節點的 IP addr : 192.168.0.106
[Sender 訊息]:: 請輸入第 2 個節點的 IP port : 51105
[Sender 訊息]:: 請輸入第 3 個節點的 IP addr : 192.168.0.106
[Sender 訊息]:: 請輸入第 3 個節點的 IP port : 51106
[Receiver 訊息]:: 收到距離更新: v 2 0 3 2

u v w x
u 0 2 5 1
v 2 0 3 2
w -1 -1 -1 -1
x -1 -1 -1 -1
```

(4) 輸入各個節點的位址和埠號

```
[Receiver 訊息]:: 收到距離更新: w 5 3 0 3

u v w x
u 0 2 5 1
v 2 0 3 2
w 5 3 0 3
x -1 -1 -1 -1
```

(5) 收到更新 Distance Vector

```
[Receiver 訊息]:: 收到距離更新: x 1 2 3 0

u v w x
u 0 2 4 1
v 2 0 3 2
w 5 3 0 3
x 1 2 3 0
```

```
[Receiver 訊息]:: 收到距離更新: w 4 3 0 3

u v w x
u 0 2 4 1
v 2 0 3 2
w 4 3 0 3
x 1 2 3 0
```

u 節點的路由表如下所示(目的/下個節點/成本): (6) 輸出節點的路由表

```
u u 0
v v 2
w x 4
x x 1
```

u 的 Distance Vector 演算法已經穩定 (7) 若 60 秒內沒有傳入封包, 則視為狀態穩定

```
[設定訊息]: 請選擇以下選項:
[設定訊息]: [1]: Network Initialization
[設定訊息]: [2]: Link cost update
[設定訊息]: [3]: Exit the program
[設定訊息]: [_]: 你的選擇: 2
[設定訊息]: 新的輸入檔的名稱: data2.txt
[更新訊息]: 原本 u 到 v 的成本由 2 變成 1
[更新訊息]: 原本 u 到 x 的成本由 1 變成 2
```

(8)

重新選擇選項(每個節點的選擇需一致, 否則結果會錯誤)

```
u v w x
```

(9)

```
u 0 1 4 2
```

```
v 2 0 3 2
```

```
w 4 3 0 3
```

```
x 1 2 3 0
```

印出自己節點成

本更新後的狀態

印出改變成本的 link

S0261001 [Java Application] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (2017年5月7日 下午12:55:36)

[Receiver 訊息]: 節點 v 收到節點 u 表示: 原本從 u 到 v 的成本由 2 變成 1 (10) 回復確認收到更新訊息

[Receiver 訊息]: 節點 v 收到節點 u 表示: 原本從 u 到 x 的成本由 1 變成 2

[Receiver 訊息]: 收到距離更新: 節點 w 表示: 原本從 w 到 v 的成本由 3 變成 5 (11) 印出收到的更新訊息

[Receiver 訊息]: 收到距離更新: w 4 5 0 3

```
u v w x
```

```
u 0 1 4 2
```

```
v 1 0 5 2
```

```
w 4 5 0 3
```

```
x 1 2 3 0
```

[Receiver 訊息]: 收到距離更新: v 1 0 5 2

```
u v w x
```

```
u 0 1 4 2
```

```
v 1 0 5 2
```

```
w 4 5 0 3
```

```
x 1 2 3 0
```

(12) 印出更新後的 Distance Vector

[Receiver 訊息]: 節點 w 收到節點 u 表示: 原本從 u 到 v 的成本由 2 變成 1

[Receiver 訊息]: 節點 w 收到節點 u 表示: 原本從 u 到 x 的成本由 1 變成 2

[Receiver 訊息]: 收到距離更新: 節點 x 表示: 原本從 x 到 u 的成本由 1 變成 2

[Receiver 訊息]: 收到距離更新: x 2 2 3 0

```
u v w x
```

```
u 0 1 4 2
```

```
v 1 0 5 2
```

```
w 4 5 0 3
```

```
x 2 2 3 0
```

[Receiver 訊息]: 收到距離更新: v 1 0 5 2

```
u v w x
```

```
u 0 1 4 2
```

```
v 1 0 5 2
```

```
w 4 5 0 3
```

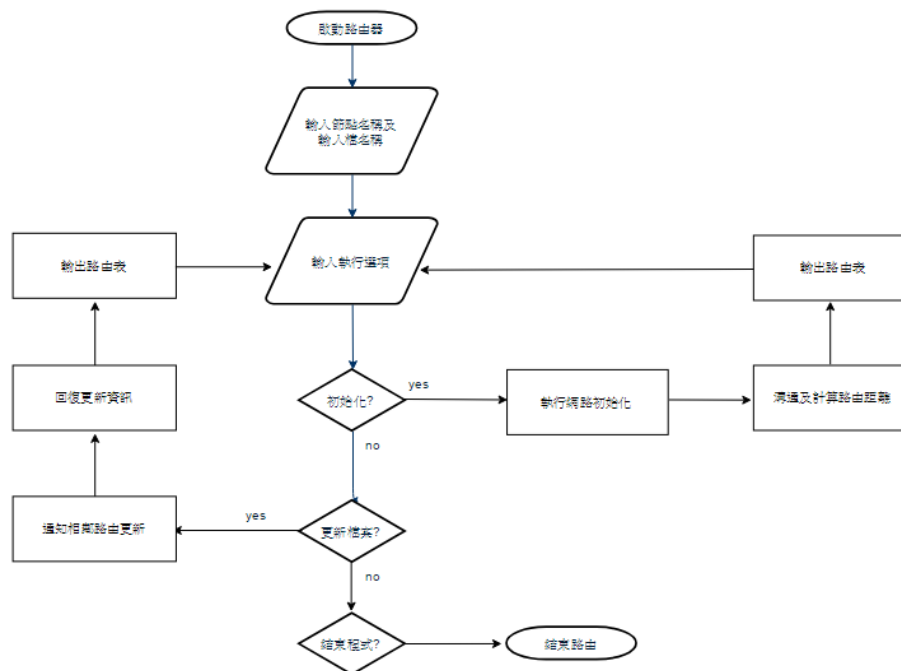
```
x 2 2 3 0
```

```

v 的 Distance Vector 演算法已經穩定
[設定訊息]:: 請選擇以下選項:
[設定訊息]:: [1]: Network Initialization
[設定訊息]:: [2]: Link cost update
[設定訊息]:: [3]: Exit the program
[設定訊息]:: [ ]: 你的選擇: 2
[設定訊息]:: 新的輸入檔的名稱: data.txt
[更新訊息]:: 原本 V 到 U 的成本由 1 變成 2
[更新訊息]:: 原本 V 到 W 的成本由 5 變成 3
    u v w x
u 0 1 4 2
v 2 0 3 2
w 4 5 0 3
x 2 2 3 0
[通知訊息]:: 節點 U 已經結束運作 (13) 節點結束程式會通知其他所有節點

```

四、本次作業程式流程說明



五、資管四 周平---本次作業的心得

本次作業在實作過程，除了統整了先前所學的 TCP/IP 課程之外，也應用了許多在《作業系統》中所學到的理論，例如：同步處理、多執行緒，從中也應證了許多在電腦網路中學到的細節知識，像是傳送的封包會被放在緩衝區中等待被接收、Socket Timeout 等部分，除此之外，我也是第一次利用 Java 語言實作一個相較複雜的系統，並且善用物件導向語言的特性幫助開發。此外，我也學習到在開發此種系統時，應在一開始將系統需求了解透徹，若是在開發的過程中才擬定開發的策略的話，可能會造成初期開發的程式在最後彙整時出現衝突、不符合需要等等狀況，因此，這項作業不僅是考驗上課所學的理論，更是考驗我們寫程式的

經驗及撰寫技巧，也因為如此，我們更加感覺到自己學習的不足。

整體而言，儘管實作這個系統的過程經歷了許多困難，我們卻也一一將其克服，從中獲益許多，並且從而了解到網路程式開發是一項相當困難的工作，未來當我們進行相同類型的程式開發時，將可以習取我們所學到的經驗，對於整體的開發流程及工作有更趨完整的概念，而不會像本次作業撰寫之初手足無措。很感謝老師上課的指導，也期許未來我們能夠在此一領域繼續精進、繼續學習。

六、資管四 蘇揮原---本次作業的心得

以往對於 java 程式語言的十座，較沒有開發到網路相關程式，所以當老師當下說出要實作出「Distance-Vector Algorithm」，當下心裡認為充滿了挑戰極困難度。一開始問自己是否能如期完成，但與其事先擔憂，不如先實際做看看。

一開始在開發上有先事想整個程式的流程，當自己想完整個流程就實際撰寫看看，一開始我先以單純的設計 client-server 端，並且也順利可以連線以及透過 Oracle Java API 取得隨機 port 與訊息傳遞，但後來一直無法成功的同時廣播到多台電腦，只能任兩台彼此互相連線。後來想不到方式，就先轉移到「檔案的輸入與 DV 表格」的撰寫，起初我先用一般手動輸入的方式去算出每個節點數與連結成本數，這部分我也有撰寫成功，但與老師的要求相去甚遠，因為在做節點更新時只能更新一個節點，無法順利將有異動過的所有節點做更新。也因為這樣，我才發現這個演算法的開發遇到許多困難，即便我已把基本的架構都撰寫完成了，但無法將這些分散完成的功能去做整合，後來去尋求於我的同學——周平。

在與他的討論的過程中，他也認為我的撰寫基本架構都十分正確，就只是插在一些地方的整合與細節去考慮到的問題，所以我們一同去討論這些狀況，病鳥解當中的解決方案。結果才發現這次的作業涵蓋了許多概念，像是我去年有修習貴系的「作業系統」課程，其中有運用到「lock」、「critical-section」的概念，這概念主要是解決每個訊息先來後到的順序以至於可以算出最後正確的 DV-table，那時候才發現到網路相關程式確實會運用到許多其他的領域的理論，才能知道這次作業的確是一項大工程，後來與周平討論後並與他組一個團隊且完成這項作業。

最後，在這次作業我發現了自己許多的不足，即便我之前有撰寫過 Java 語言的經驗，但沒有時做過網路的開發，還有一些 function、API 的運用，也感謝老師出了這項作業，讓我在藉此增進自己的能力並正視自己的問題點，而我也從同學身上學到了不少知識及專業，即便最後在程式的開發上負責得並不多，但也同時警惕著自己，若未來要走向這條路，還有很多需要自己學習之處。

七、本次作業撰寫過程中所遇之問題與解決之道:

1. 在 Socket 連線上如何進行同步廣播訊息到其他電腦，使每台電腦可以做各自資料上的更新。

Sol: 我們最後實作的方法是利用兩個一維陣列儲存位址及埠號資訊，幫有需要傳送訊息的時候，即透過查詢這些陣列，找出對應的位址及埠號，藉此將欲傳送的資料傳送到目的地端。接收方則只需負責讀取資料，並於本機處理更新即可。

2. 於第一階段，將每台電腦各自的資料初始化之後，依常理來說，如何確保初始化完的資料是穩定的？

Sol: 我們使用的方法是給定一個 Timeout 時間，若接收端在 Timeout 時間之內沒有再收到其他訊息，則代表每個節點的發送端已經進入閒置狀態，也就代表沒有新的成本更新訊息，藉此我們可以得知資料已經達成穩定。

3. 執行第二階段時，發現一旦更新只能更新一個節點，無法將全部檔案中異動過的值全部更新。

Sol: 此時，我們會利用宣告一個陣列空間將所有有異動過的資料存放進去，再與每台電腦的狀況做一一比對，一旦有不同則會取代舊有值，以做更新。

4. 在執行第二階段，一旦讀取到最新的 txt 檔並判斷有異動過的值時，雖然的確有互相傳送訊息更新，但發現最後更新完的結果有錯誤。(以手寫方式檢驗過，發現程式執行有誤)

Sol: 經由檢驗之後，我們發現當接收端收到更新資訊，並且更新了資料時，此時發送端可能尚未取得 CPU 的使用資源，因此當接收端再收到其他訊息時，重複檢查更新結果會產生「沒有改變」的檢查錯誤。因此，此部份我們已利用 lock 來解決 critical section 產生的同步問題，已達成正確的更新檢查工作。

5. 傳送訊息的種類有許多種，我們發現無法僅用一個接收函式處理所有種類的訊息接收，若是紀錄太多的狀態資訊又會過於複雜

Sol: 此部分，我們是在傳送訊息的開頭加上標記(header 資料)，如此一來，接收端收到資料時，即可透過開頭的 header 資訊，得知此訊息的種類，並且進一步將此資訊交由對應的函式來處理，藉此達成處理多種不同種類之訊息。

6. 當傳送的訊息包含中文字元時，接收端收到的結果會和傳送端不同

Sol: 經由研究，我們得知在 Java 語言中，計算字串長度會將中文字元視為是一個字元的長度，然而，其編碼占用的位元長度卻和英文不同。因此，最後我們在傳輸和收取資料部分，是以 Byte 長度為準，藉以精準的使用中文傳輸資訊。