1. Confidentiality is just one attribute of secure communication. Describe at least three others.

2. Refer to the source code for a simple Web server whose listing begins on page 2 of this examination.
   (a) The program waits for a client to send it an HTTP request. What does the program expect to find in the HTTP request?
   (b) How will the program respond to a client’s request? Describe the format of the message that this server returns to the client.
   (c) Suggest a way of improving this program. What would you try next?

3. Figures 1 and 2 describe the same network. Figure 3 shows the table that Dijsktra’s algorithm produced when used to find the shortest paths from node a to all other nodes in the network.
   (a) What information does the column labeled $N'$ contain?
   (b) What is the length of the shortest path from node a to node f?
   (c) What is the shortest path that connects node a to node f? Explain how you are able to read this sequence of nodes from the table.
   (d) Suppose that a router at node a contains a forwarding table with entries for all nodes b–h. What are the entries in this table?

4. Refer to the excerpt of a trace of the distance vector algorithm on page 5 of this examination. The same program that we examined in the laboratory and in the classroom produced this trace from the same example network. The trace shows the initialization of the network’s four nodes and the first few updates of node D0.
   (a) Describe the meaning of the vector $(7 \infty 2 0)$ that node D0 receives from node D3 at time $time = 4.104$.
   (b) What do the numbers in the $3 \times 3$ table that node D0 maintains signifies?
(c) The program changes the value of the element that is in the second row and the third column of the table. At \( \text{time} = 2.407 \), the value of that element was infinity. After the update at \( \text{time} = 4.104 \), its value is nine. Explain how the program computed the new value.

(d) After the update at \( \text{time} = 2.407 \), node D0 sent packets to the other three nodes. After the update at \( \text{time} = 4.104 \), it sends no packets. Why not?

5. In our exploration of the network layer, we studied routing algorithms. We used graphs to model the topology of networks through which we routed packets.

In our exploration of the transport layer, we studied protocols for reliable data transfer. There, we used graphs to model the logic of sending and receiving processes. More particularly, we used graphs to model finite state machines.

A finite state machine is a set of states, a set of possible inputs, a mapping that associates each combination of state and input with a new state, and a designation of a starting state.

(a) What part of the notation in figure 4 denotes the states?
(b) What part of the notation in the figure denotes inputs?
(c) What identifies the starting state in each of the two finite state machines depicted in figure 4?
(d) These finite state machines model \textit{rdt3.0}, the final stop-and-wait, alternating bit reliable data transfer protocol that the authors presented to us in Chapter 2. Which finite state machine models the sender? Which models the receiver?
(e) The diagram describes the transition mappings in both finite state machines. One could also specify the mapping in a finite state machine with a table. Show that you understand transitions between states in a finite state machine by constructing a table for the simpler of the two finite state machines.

Label each row of your table with a state. Label each column with an input. Each entry of your table will be either one of the states depicted in the diagram or a special error state.

(f) The authors our textbook first gave us a very simple model of a sender and receiver, then developed a model of reliable data transfer by adding acknowledgements, sequence numbers, and a timer. How can this stop-and-wait, alternating bit protocol be improved?

Simple Web Server
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.StringTokenizer;

public class SimpleWebServer {

    public static void main(String[] arguments) throws Exception {
        // Listen for client's "knock"
        ServerSocket listenSocket = new ServerSocket(PORTNUMBER);

        // Open "door" (TCP connection) to client
        Socket connectionSocket = listenSocket.accept();

        // Create means of receiving requests from clients
        InputStream is = connectionSocket.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader inFromClient = new BufferedReader(isr);

        // Create means of sending response (HTML files) to clients
        OutputStream os = connectionSocket.getOutputStream();
        DataOutputStream outToClient = new DataOutputStream(os);

        // Receive a request from client
        String requestMessageLine = inFromClient.readLine();

        // Parse request
        StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);

        // first word in request should be "GET"
        if (tokenizedLine.nextToken().equals("GET")) {
            // second word in request should be name of file
            String fileName = tokenizedLine.nextToken();

            // remove slash from front of name of file
            if (fileName.startsWith("/")) {
                fileName = fileName.substring(1);
            }
        }
    }
}
// measure size of requested file
File file = new File( fileName ) ;
    int numOfBytes = (int) file.length() ;

// Create buffer for contents of requested file
FileInputStream inFile = new FileInputStream( fileName ) ;
    byte [] fileInBytes = new byte[ numOfBytes ] ;

// Copy contents of requested file into buffer
inFile . read( fileInBytes ) ;

// Write first line of response
outToClient . writeBytes( "HTTP/1.0 200 Document Follows \r\n" ) ;

// Identify image files
if( fileName . endsWith( " . jpg " ) ) {
    outToClient . writeBytes( "Content-Type: image/jpeg \r\n" ) ;
} // if

// Return a line to the client that identifies length of
// the file that this server is sending to the client
outToClient . writeBytes( "Content-Length: \n" + numOfBytes + " \r\n" ) ;

// Send a blank line to client
outToClient . writeBytes( "\r\n" ) ;

// Send requested file to client
outToClient . write( fileInBytes , 0 , numOfBytes ) ;

// All done! Clean up by closing socket
connectionSocket . close() ;
} // if
else { System . out . println( "Bad Request Message" ) ;
} // else

} // main( String [] )

private static final int PORT_NUMBER = 1729 ;

} // SimpleWebServer
Partial trace of distance-vector algorithm

Time: 0.000
At time t=0.000, rtinit0() called.

\[
\begin{array}{c|ccc}
D0 & 1 & 2 & 3 \\
\hline
\text{via} & & & \\
1 & 1 & \infty & \infty \\
\text{dest} & 2 & \infty & 3 \\
3 & \infty & \infty & 7 \\
\end{array}
\]

At time t=0.000, node 0 sends packet to node 1 with: 0 1 3 7
At time t=0.000, node 0 sends packet to node 2 with: 0 1 3 7
At time t=0.000, node 0 sends packet to node 3 with: 0 1 3 7

At time t=0.000, rtinit1() called

\[
\begin{array}{c|c}
D1 & 0 \\
\hline
\text{via} & 2 \\
0 & 1 & \infty \\
\text{dest} & 2 & \infty \\
3 & \infty & \infty \\
\end{array}
\]

At time t=0.000, node 1 sends packet to node 0 with: 1 0 1 \infty
At time t=0.000, node 1 sends packet to node 2 with: 1 0 1 \infty

At time t=0.000, rtinit2() called

\[
\begin{array}{c|ccc}
D2 & 0 & 1 & 3 \\
\hline
\text{via} & & & \\
0 & 3 & \infty & \infty \\
\text{dest} & 1 & \infty & 1 \\
3 & \infty & \infty & 2 \\
\end{array}
\]

At time t=0.000, node 2 sends packet to node 0 with: 3 1 0 2
At time t=0.000, node 2 sends packet to node 1 with: 3 1 0 2
At time t=0.000, node 2 sends packet to node 3 with: 3 1 0 2

At time t=0.000, rtinit3() called.

\[
\begin{array}{c|c}
D3 & 0 \\
\hline
\text{via} & 2 \\
0 & 7 & \infty \\
\text{dest} & 1 & \infty \\
2 & \infty & 2 \\
\end{array}
\]

At time t=0.000, node 3 sends packet to node 0 with: 7 \infty 2 0
At time t=0.000, node 3 sends packet to node 2 with: 7 \infty 2 0

Time: 0.992
MAIN: rcv event, t=0.992, at src: 1, dest: 0, contents: 1 0 1 \infty
At time t=0.992, rtupdate0() called, and node 0 receives a packet from node 1
At time $t=0.992$, node 0 sends packet to node 1 with: 0 1 2 7
At time $t=0.992$, node 0 sends packet to node 2 with: 0 1 2 7
At time $t=0.992$, node 0 sends packet to node 3 with: 0 1 2 7

**Time: 2.407**

MAIN: rcv event, $t=2.407$, at 0 src: 2, dest: 0, contents: 3 1 0 2
At time $t=2.407$, rtupdate0() called, and node 0 receives a packet from node 2

<table>
<thead>
<tr>
<th>D0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>dest 2</td>
<td>2</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>7</td>
</tr>
</tbody>
</table>

At time $t=2.407$, node 0 sends packet to node 1 with: 0 1 2 5
At time $t=2.407$, node 0 sends packet to node 2 with: 0 1 2 5
At time $t=2.407$, node 0 sends packet to node 3 with: 0 1 2 5

**Time: 4.104**

MAIN: rcv event, $t=4.104$, at 0 src: 3, dest: 0, contents: 7 $\infty$ 2 0
At time $t=4.104$, rtupdate0() called, and node 0 receives a packet from node 3

<table>
<thead>
<tr>
<th>D0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>$\infty$</td>
</tr>
<tr>
<td>dest 2</td>
<td>2</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>3</td>
<td>$\infty$</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Figure 1: Network (not drawn to scale).

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>∞</td>
<td>∞</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>f</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>∞</td>
</tr>
<tr>
<td>g</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
<td>1</td>
<td>∞</td>
<td>6</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>h</td>
<td>∞</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2: Network.

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>1,a</td>
<td>4,a</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ab</td>
<td>3,b</td>
<td>10,b</td>
<td>∞</td>
<td>∞</td>
<td>5,b</td>
<td>3,b</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>abc</td>
<td>4,c</td>
<td>6,c</td>
<td>∞</td>
<td>5,b</td>
<td>3,b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>abch</td>
<td>4,c</td>
<td>6,c</td>
<td>∞</td>
<td>5,b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>abchd</td>
<td>5,d</td>
<td>7,d</td>
<td>5,b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>abchde</td>
<td>6,e</td>
<td>5,b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>abchdeg</td>
<td>6,e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Table produced by Dijkstra's algorithm.
Figure 4: Finite state machines to model reliable data transfer.