



# GALGOTIAS UNIVERSITY

## LAB MANUAL

### DATA COMMUNICATION AND NETWORKING

---

**Name of School:** SCHOOL OF COMPUTING SCIENCE &  
ENGINEERING

**Department:** COMPUTER SCIENCE & ENGINEERING

**Year:** 2022-2023



<b>SUBJECT</b>	<b>Data Communication &amp; Networking</b>	<b>PROGRAMME</b>	<b>B. Tech.</b>
<b>SUBJECT CODE</b>	<b>BCSE2370</b>	<b>SEMESTER</b>	<b>3</b>
<b>CREDITS</b>	<b>1</b>	<b>DURATION OF SEMESTER</b>	<b>13 Weeks</b>
<b>PREREQUISITE SUBJECTS</b>	<b>C/C++</b>	<b>SESSION DURATION</b>	<b>2 Hrs per Week</b>

### **Vision**

"To be recognized globally as a premier School of Computing Science and Engineering for imparting quality and value based education within a multi-disciplinary and collaborative research based environment."

### **Mission**

**The mission of the school is to:**

**M1:** Develop a strong foundation in fundamentals of computing science and engineering with responsiveness towards emerging technologies.

**M2:** Establish state-of-the-art facilities and adopt education 4.0 practices to analyze, develop, test and deploy sustainable ethical IT solutions by involving multiple stakeholders.

**M3:** Foster multidisciplinary collaborative research in association with academia and industry through focused research groups, Centre of Excellence, and Industry Oriented R&D Labs.

## **PROGRAM EDUCATIONAL OBJECTIVES**

**The Graduates of Computer Science and Engineering shall:**

**PEO1:** be engaged with leading Global Software Services and Product development companies handling projects in cutting edge technologies.

**PEO2:** serve in technical or managerial roles at Government firms, Corporates and contributing to the society as successful entrepreneurs through startup.

**PEO3:** undertake higher education, research or academia at institutions of transnational reputation.

## **PROGRAMME SPECIFIC OUTCOME (PSO):**

**The students of Computer Science and Engineering shall:**

**PSO1:** Have the ability to work with emerging technologies in computing requisite to Industry 4.0.

**PSO2:** Demonstrate Engineering Practice learned through industry internship and research project to solve live problems in various domains.

## **PROGRAMME OUTCOME (PO):**

**PO1 Computing Science knowledge:** Apply the knowledge of mathematics, statistics, computing science and information science fundamentals to the solution of complex computer application problems.

**PO2 Problem analysis:** Identify, formulate, review research literature, and analyze complex computing science problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and computer sciences.

**PO3 Design/development of solutions:** Design solutions for complex computing problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4 Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5 Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern computing science and IT tools including prediction and modeling to complex computing activities with an understanding of the limitations.

**PO6 IT specialist and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional computing science and information science practice.

**PO7 Environment and sustainability:** Understand the impact of the professional computing science solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8 Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the computing science practice.

**PO9 Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10 Communication:** Communicate effectively on complex engineering activities with the IT analyst community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11 Project management and finance:** Demonstrate knowledge and understanding of the computing science and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12 Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## List of Programs

	Programs
1	Implement bit stuffing
2	Implement bit de-stuffing
3	Write a program for hamming code generation for error detection and correction
4	Implement Cyclic Redundancy Check (CRC)
5	Write a program for congestion control using the leaky bucket algorithm
6	Implement Dijkstra's algorithm to compute a shortest path through graph.
7	Take a 64-bit plain text and encrypt the same using DES algorithm
8	Using RSA algorithm encrypt a text data and decrypt the same.
9	Implementation of the link state routing protocols.
10	Implementation of LZW compression and decompression algorithms.

### Value Added Programs:

1. Implementation of Client-Server Communication Using TCP
2. Implementation of Client – Server Communication using RPC
3. Implementation of Distance Vector algorithms.

### Course Outcomes

Upon successful completion of this course, students will be able to

<b>CO1</b>	Understand the different networking sub-systems and their functions in a telecommunication system.
<b>CO2</b>	Understand and configure the different types of network topologies and protocols.
<b>CO3</b>	Understand the different protocols layers of the OSI model.
<b>CO4</b>	Examine and analyze the network-layer concepts like Network-Layer services – Routing -IP protocol -IP addressing
<b>CO5</b>	Examine and analyze the different link-layer and local area network concepts like Link-Layer services –Ethernet -Token Ring -Error detection and correction -ARP protocol
<b>CO6</b>	Gain understanding of latest trends and research areas in the course

## CO-PO-PSO MAPPING:

CO/PO Mapping														
(S/M/W or 3/2/1 indicates strength of correlation) S/3-Strong, M/2-Medium, L/1-Low														
CO's	Programme Outcomes (POs) / Programme Specific Outcome (PSO)													
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	2		1	1	3				2	2		3	1	
CO2	2								1	1		2		
CO3		1		1					1			1	1	2
CO4														
CO5		1												2

## Continuous Assessment Pattern

Practical IA	ETE	Total
100	0	100

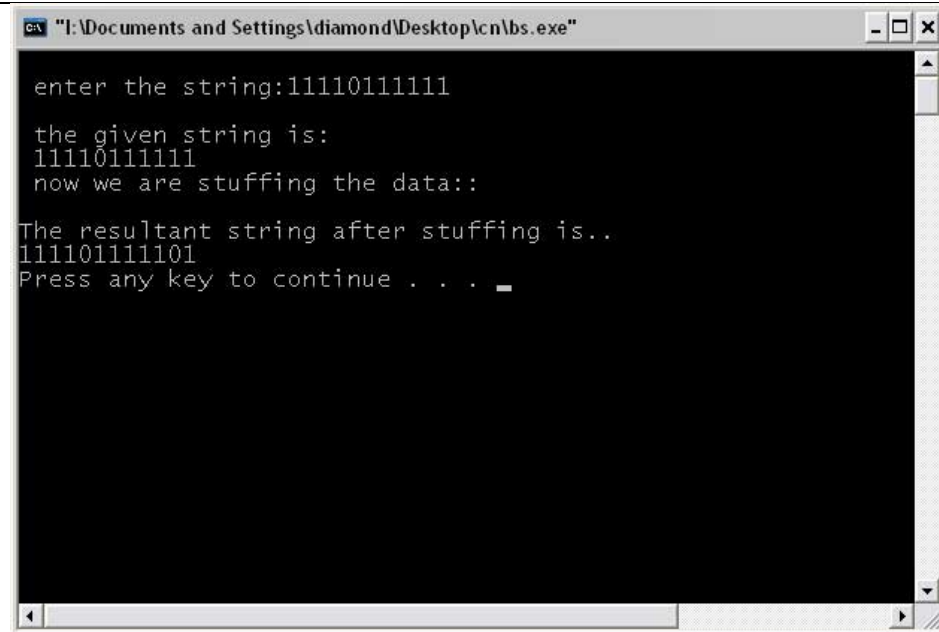
## Rubrics for Practical IA

S. No.	Rubrics - Parts	Marks
1	Performance	2
2	Result	3
3	File	2
4	Viva	3
Total		10

Experiment No:1	
Title	Implement bit stuffing and de-stuffing.
Algorithm/ Theory	<p>In data transmission and telecommunication, bit stuffing (also known—uncommonly—as positive justification) is the insertion of non-information bits into data. Stuffed bits should not be confused with overhead bits.</p> <p>Bit stuffing is used for various purposes, such as for bringing bit streams that do not necessarily have the same or rationally related bit rates up to a common rate, or to fill buffers or frames. The location of the stuffing bits is communicated to the receiving end of the data link, where these extra bits are removed to return the bit streams to their original bit rates or form. Bit stuffing may be used to synchronize several channels before multiplexing or to rate-match two single channels to each other.</p> <p>Applications include Plesiochronous Digital Hierarchy and Synchronous Digital Hierarchy.</p> <p><b>Source Code</b></p> <pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt; void stuff(char str[40]); main() {     char str[40];     int i;     printf("\n enter the string:");     scanf("%s", &amp;str);     printf("\n the given string is:");     printf("\n %s", str);     stuff(str); }  void stuff(char str[40]) {     int i, j, k = 0, l, n, z;     printf("\n now we are stuffing the data::\n");     n = strlen(str);     for (i = 0; i &lt; n; i++)     {         if (str[i] == '1')         {             k = 1;             for (l = i + 1; l &lt;= i + 5; l++)             {                 if (str[l] == '1')                     k++;             }             else                 break;         }         if (k == 6)         {             i = i + 6;             z = n + 1;             for (j = z; j &gt;= i; j--) </pre>

```
    {  
        str[j] = str[j - 1];  
    }  
    str[j] = '0';  
}  
}  
}  
printf("\nThe resultant string after stuffing is..\n");  
printf("%s\n", str);  
}
```

**Sample  
Output**



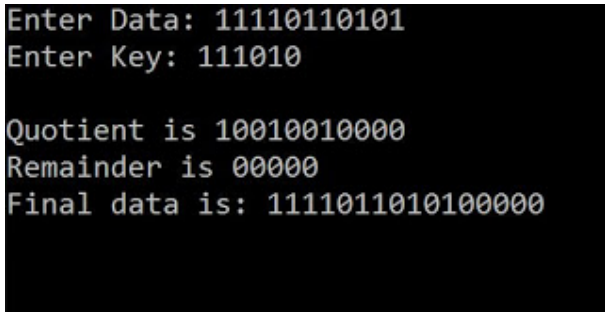
```
ca "I:\Documents and Settings\diamond\Desktop\cn\bs.exe"  
  
enter the string:11110111111  
  
the given string is:  
11110111111  
now we are stuffing the data::  
  
The resultant string after stuffing is..  
111101111101  
Press any key to continue . . . _
```



Experiment No:3	
<b>Title</b>	<b>Write a program for hamming code generation for error detection and correction.</b>
<b>Algorithm/ Theory</b>	<p>In telecommunication, a Hamming code is a linear error-correcting code named after its inventor, Richard Hamming. Hamming codes can detect up to two simultaneous bit errors, and correct single-bit errors; thus, reliable communication is possible when the Hamming distance between the transmitted and received bit patterns is less than or equal to one. By contrast, the simple parity code cannot correct errors, and can only detect an odd number of errors.</p> <p><b>Source Code</b></p> <pre> #include&lt;stdlib.h&gt; #include&lt;stdio.h&gt; char data[5]; int encoded[8], edata[7], syndrome[3]; int hmatrix[3][7]= { 1,0,0,0,1,1,1, 0,1,0,1,0,1,1, 0,0,1,1,1,0,1}; char gmatrix[4][8]={ "0111000", "1010100", "1100010", "1110001"}; int main() { int i,j; system("clear"); printf("Hamming Code --- Encoding\n"); printf("Enter 4 bit data : "); scanf("%s",data); printf("Generator Matrix\n"); for(i=0;i&lt;4;i++) printf("\t %s \n",gmatrix[i]); printf("Encoded Data : "); for(i=0;i&lt;7;i++) { for(j=0;j&lt;4;j++) encoded[i]+=((data[j]- '0')*(gmatrix[j][i]- '0')); encoded[i]=encoded[i]%2; printf("%d",encoded[i]); } printf("\nHamming code --- Decoding\n"); printf("Enter Encoded bits as received : "); for(i=0;i&lt;7;i++) scanf("%d",&amp;edata[i]); for(i=0;i&lt;3;i++){ for(j=0;j&lt;7;j++) syndrome[i]=syndrome[i]+(edata[j]*hmatrix[i][j]); syndrome[i]=syndrome[i]%2;} for(j=0;j&lt;7;j++) if ((syndrome[0]==hmatrix[0][j])&amp;&amp;(syndrome[1]==hmatrix[1][j])&amp;&amp; (syndrome[2]==hmatrix[2][j])) break; if(j==7) printf("Data is error free!!\n"); else { printf("Error received at bit number %d of the data\n",j+1); } } </pre>

	<pre> edata[j]!=edata[j]; printf("The Correct data Should be : "); for(i=0;i&lt;7;i++) printf(" %d ",edata[i]); }}</pre>
<b>Sample Output</b>	<pre> Hamming Code --- Encoding Enter 4 bit data : 1001 Generator Matrix 0111000 1010100 1100010 1110001 Encoded Data : 1001001 Hamming code --- Decoding Enter Encoded bits as received : 1 0 0 1 0 0 1 Data is error free!! Hamming Code --- Encoding Enter 4 bit data : 1011 Generator Matrix 0111000 1010100 1100010 1110001 Encoded Data : 0101011 Hamming code --- Decoding Enter Encoded bits as received : 0 1 0 1 0 1 0 Error received at bit number 7 of the data The Correct data Should be : 0 1 0 1 0 1 1</pre>

Experiment No:4	
<b>Title</b>	<b>Implement cyclic redundancy check CRC.</b>
<b>Algorithm/ Theory</b>	<p>A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.</p> <p><b>Source Code</b></p> <pre> #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;string.h&gt; void main() { int i,j,keylen,msglen; char input[100], key[30],temp[30],quot[100],rem[30],key1[30]; clrscr(); printf("Enter Data: "); gets(input); printf("Enter Key: "); gets(key); keylen=strlen(key); msglen=strlen(input); strcpy(key1,key); for(i=0;i&lt;keylen-1;i++) { input[msglen+i]='0'; } for(i=0;i&lt;keylen;i++) temp[i]=input[i]; for(i=0;i&lt;msglen;i++) { quot[i]=temp[0]; if(quot[i]=='0') for(j=0;j&lt;keylen;j++) key[j]='0'; else for(j=0;j&lt;keylen;j++) key[j]=key1[j]; for(j=keylen-1;j&gt;0;j--) { if(temp[j]==key[j]) rem[j-1]='0'; else rem[j-1]='1'; } rem[keylen-1]=input[i+keylen]; strcpy(temp,rem); } strcpy(rem,temp); printf("\nQuotient is "); for(i=0;i&lt;msglen;i++) </pre>

	<pre>printf("%c",quot[i]); printf("\nRemainder is "); for(i=0;i&lt;keylen-1;i++) printf("%c",rem[i]); printf("\nFinal data is: "); for(i=0;i&lt;msglen;i++) printf("%c",input[i]); for(i=0;i&lt;keylen-1;i++) printf("%c",rem[i]); getch(); }</pre>
<b>Sample Output</b>	

Experiment No:5	
<b>Title</b>	<b>Write a program for congestion control using the leaky bucket algorithm.</b>
<b>Algorithm/ Theory</b>	<p>The Leaky Bucket Algorithm is an algorithm used in packet switched computer networks and telecommunications networks. It can be used to check that data transmissions, in the form of packets, conform to defined limits on bandwidth and burstiness (a measure of the unevenness or variations in the traffic flow). It can also be used as a scheduling algorithm to determine the timing of transmissions that will comply with the limits set for the bandwidth and burstiness: see network scheduler. The leaky bucket algorithm is also used in leaky bucket counters, e.g. to detect when the average or peak rate of random or stochastic events or stochastic processes exceed defined limits.</p> <p><b>C. SOURCE CODE</b></p> <pre> #include&lt;stdio.h&gt; #include&lt;math.h&gt; #include&lt;stdlib.h&gt; void main() { int packets[8],i,j,clk,b_size,o_rate,i_rate,p_sz_rm=0,p_sz,p_time; clrscr(); for(i=0;i&lt;5;++i) { packets[i]=rand()%10; if(packets[i]==0) --i; } printf("Enter output rate:"); scanf("%d",&amp;o_rate); printf("\nEnter bucket size:"); scanf("%d",&amp;b_size); for(i=0;i&lt;5;++i) { if((packets[i]+p_sz_rm) &gt; b_size) { if(packets[i]&gt;b_size) printf("\nIncoming packet size:%d greater than bucket capacity\n",packets[i]); else printf("Bucket size exceeded\n"); } else { p_sz=packets[i]; p_sz_rm+=p_sz; printf("\n----- \n"); printf("Incoming packet:%d",p_sz); printf("\nTransmission left:%d\n",p_sz_rm); p_time=rand()%10; printf("Next packet will come at %d",p_time); for(clk=0;clk&lt;p_time&amp;&amp; p_sz_rm&gt;0;++clk) { printf("\nTime left %d---No packets to transmit!!\n",p_time-clk); sleep(1); if(p_sz_rm) { printf("Transmitted\n"); if(p_sz_rm&lt;o_rate) </pre>

	<pre> p_sz_rm=0; else p_sz_rm==o_rate; printf("Bytes remaining:%d\n",p_sz_rm); } else printf("No packets to transmit\n"); } } } } getch(); } </pre>
<b>Sample Output</b>	<pre> Enter output rate:2 Enter bucket size:10 ----- Incoming packet:6 Transmission left:6 Next packet will come at 7 Time left 7-----No packets to transmit!! Transmitted Bytes remaining:4 Time left 6-----No packets to transmit!! Transmitted Bytes remaining:2 Time left 5-----No packets to transmit!! Transmitted Bytes remaining:0 ----- Incoming packet:0 Transmission left:0 Next packet will come at 5 ----- Incoming packet:2 Transmission left:2 Next packet will come at 5 Time left 5-----No packets to transmit!! Transmitted Bytes remaining:0 ----- Incoming packet:0 Transmission left:0 Next packet will come at 8 ----- Incoming packet:6 Transmission left:6 Next packet will come at 6 Time left 6-----No packets to transmit!! Transmitted Bytes remaining:4 Time left 5-----No packets to transmit!! Transmitted Bytes remaining:2 Time left 4-----No packets to transmit!! </pre>

	Transmitted Bytes remaining:0
--	----------------------------------

Experiment No:6	
<b>Title</b>	Implement dijkstra's algorithm to compute a shortest path through graph.
<b>Algorithm/ Theory</b>	<p>Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.</p> <p><b>Source Code</b></p> <pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #define INFINITY 9999  int main() {     int vertexCount, **edgeLength, **res;     int i, j, k;     printf("Enter the no of vertices:");     scanf("%d", &amp;vertexCount);     edgeLength = (int **)calloc(sizeof(int), vertexCount);     res = (int **)calloc(sizeof(int), vertexCount);     for (i = 0; i &lt; vertexCount; i++) {         edgeLength[i] = (int *)calloc(sizeof(int), vertexCount);         res[i] = (int *)calloc(sizeof(int), vertexCount);     }      /* Adjacency Matrix to store Cost of the edges */     for (i = 0; i &lt; vertexCount; i++) {         for (j = 0; j &lt; vertexCount; j++) {             printf("Edge weight %d to %d(0 if no edge):", i + 1, j + 1);             scanf("%d", &amp;edgeLength[i][j]);             if (edgeLength[i][j] == 0) {                 res[i][j] = INFINITY;             } else {                 res[i][j] = edgeLength[i][j];             }         }     }      printf("Adjacent matrix for edge weights:\n");     for (i = 0; i &lt; vertexCount; i++) {         for (j = 0; j &lt; vertexCount; j++) {             printf("%3d", edgeLength[i][j]);         }         printf("\n");     }      /* Calculate shortest path from each vertex to every other vertices */     for (i = 0; i &lt; vertexCount; i++) {         for (j = 0; j &lt; vertexCount; j++) {             for (k = 0; k &lt; vertexCount; k++) {                 if (res[j][k] &gt; res[j][i] + res[i][k]) {                     res[j][k] = res[j][i] + res[i][k];                 }             }         }     } </pre>



	<pre>         }     }     printf("\nShortest path between vertices\n");     for (i = 0; i &lt; vertexCount; i++) {         for (j = 0; j &lt; vertexCount; j++) {             if (res[i][j] == INFINITY)                 printf("%3d", 0);             else                 printf("%3d", res[i][j]);         }         printf("\n");     }     return 0; } } </pre>
<b>Sample Output</b>	<p>Enter the no of vertices:4</p> <p>Edge weight 1 to 1(0 if no edge):0  Edge weight 1 to 2(0 if no edge):0  Edge weight 1 to 3(0 if no edge):9  Edge weight 1 to 4(0 if no edge):4  Edge weight 2 to 1(0 if no edge):5  Edge weight 2 to 2(0 if no edge):0  Edge weight 2 to 3(0 if no edge):0  Edge weight 2 to 4(0 if no edge):0  Edge weight 3 to 1(0 if no edge):10  Edge weight 3 to 2(0 if no edge):6  Edge weight 3 to 3(0 if no edge):0  Edge weight 3 to 4(0 if no edge):8  Edge weight 4 to 1(0 if no edge):0  Edge weight 4 to 2(0 if no edge):7  Edge weight 4 to 3(0 if no edge):0  Edge weight 4 to 4(0 if no edge):0</p> <p>Adjacent matrix for edge weights:</p> <pre> 0 0 9 4 5 0 0 0 10 6 0 8 0 7 0 0 </pre> <p>Shortest path between vertices</p> <pre> 16 11 9 4 5 16 14 9 10 6 19 8 12 7 21 16 </pre>

<b>Experiment No:7</b>	
<b>Title</b>	<b>Implementation of LZW compression and decompression algorithms.</b>
<b>Algorithm/Theory</b>	<p>To compress or decompress an input array using the Lempel-Ziv-Welch (LZW) algorithm. The LZW algorithm is a lossless data compression algorithm created by Terry Welch in 1984. This algorithm represents an improved version of the LZ78 algorithm created by Abraham Lempel and Jacob Ziv in 1978.</p> <p>The idea of the compression algorithm is the following: as the input data is being processed, a dictionary keeps a correspondence between the longest encountered words and a list of code values. The words are replaced by their corresponding codes and so the input file is compressed. Therefore, the efficiency of the algorithm increases as the number of long, repetitive words in the input data increases.</p> <p>Either when using the compression or the decompression methods, the elements of the input array must be of type <b>unsigned char</b>, which is also the type of the resulting array's elements.</p> <p>The following example generates a sample array of N random letters (from A to Z) and compresses it. The compressed array is then decompressed to see if the sample array is identical to the uncompressed array. The size of the compressed array is also displayed, to prove the efficiency of the LZW algorithm.</p> <p><b>Source Code</b></p> <pre>#include &lt;codecogs/computing/io/compression/lzw.h&gt; #include &lt;iostream&gt;  // the number of characters to generate in the sample array #define N 10000  using namespace Computing::IO::Compression;  int main() {     // initialize random seed     srand(time(0));      // generate an array of N random letters     std::vector&lt;unsigned char&gt; sample;     for (int i = 0; i &lt; N; ++i)         sample.push_back('A' + rand() % ('Z' - 'A' + 1));      // compress the sample array     std::vector&lt;unsigned char&gt; compressed = LZW::compress(sample);      // decompress the compressed array     std::vector&lt;unsigned char&gt; uncompressed = LZW::decompress(compressed);      // compare the sizes of the compressed and uncompressed arrays     std::cout &lt;&lt; "    Size of the sample array: " &lt;&lt; N &lt;&lt; std::endl;     std::cout &lt;&lt; "    Size of the compressed array: " &lt;&lt; compressed.size() &lt;&lt;     std::endl;     std::cout &lt;&lt; "Size of the uncompressed array: " &lt;&lt; uncompressed.size() &lt;&lt;     std::endl;</pre>

	<pre> std::cout &lt;&lt; std::endl;  // test if the sample and the uncompressed arrays are identical // this proves that the LZW compression algorithm does not affect the initial data bool identical = (N == uncompressed.size()); for (size_t i = 0; identical &amp;&amp; i &lt; uncompressed.size(); ++i)     if (sample[i] != uncompressed[i])         identical = false;  if (identical)     std::cout &lt;&lt; "The sample and uncompressed arrays are identical." &lt;&lt; std::endl; else     std::cout &lt;&lt; "Error! The sample and uncompressed arrays are NOT identical." &lt;&lt; std::endl;  return 0; } </pre>
<b>Sample Output</b>	<p>Size of the sample array: 10000  Size of the compressed array: 7621  Size of the uncompressed array: 10000</p> <p>The sample and uncompressed arrays are identical.</p>

Experiment No:8	
Title	Using RSA algorithm encrypt a text data and decrypt the same
Algorithm/ Theory	<p>RSA is a cryptosystem, which is known as one of the first practicable public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician, had developed an equivalent system in 1973, but it wasn't declassified until 1997.</p> <pre> #include&lt;stdio.h&gt; #include&lt;conio.h&gt; int phi,M,n,e,d,C,FLAG; int check() { int i; for(i=3;e%i==0 &amp;&amp; phi%i==0;i+2) { FLAG = 1; return; } FLAG = 0; } void encrypt() { int i; C = 1; for(i=0;i&lt; e;i++) C=C*M%n; C = C%n; printf("\n\tEncrypted keyword : %d",C); } void decrypt() { int i; M = 1; for(i=0;i&lt; d;i++) M=M*C%n; M = M%n; printf("\n\tDecrypted keyword : %d",M); } void main() { int p,q,s; clrscr(); printf("Enter Two Relatively Prime Numbers\t: "); scanf("%d%d",&amp;p,&amp;q); n = p*q; phi=(p-1)*(q-1); printf("\n\tF(n) phi value\t= %d",phi); do { </pre>

	<pre> printf("\n\nEnter e which is prime number and less than phi \t: ",n); scanf("%d",&amp;e); check(); }while(FLAG==1); d = 1; do { s = (d*e)%phi; d++; }while(s!=1); d = d-1; printf("\n\tPublic Key\t: { %d,%d}",e,n); printf("\n\tPrivate Key\t: { %d,%d}",d,n); printf("\n\nEnter The Plain Text\t: "); scanf("%d",&amp;M); encrypt(); printf("\n\n the Cipher text\t: "); scanf("%d",&amp;C); decrypt(); getch(); } </pre>
<b>Sample Output</b>	<p>Write two distinct Prime number separated by space:7 17</p> <p>Enter a prime number 'e' as <math>GCD(e,(P-1)*(Q-1)) : 5</math></p> <p>public key( e , n ) : ( 5 119 )</p> <p>Private Key( d,n ) : ( 77 119 )</p> <p>Enter the Plain Text: the quick brown fox runs over the lazy dog</p> <p>114 83 33 2 78 87 56 29 116 2 98 88 76 0 94 2 51 76 1 2 88 87 94 47 2 76 118 33 88 2 114 83 33 2 75 20 5 32 2 53 76 52</p> <p>The cipher text: the quick bro n fo ☺ runs over the la♥☹ dog.</p>

Experiment No:9	
Title	Implementation Link State Routing.
Algorithm/ Theory	<p>A link-state routing protocol is one of the two main classes of <u>routing protocols</u> used in <u>packet switching</u> networks for <u>computer communications</u> (the other is the <u>distance-vector routing protocol</u>). Examples of link-state routing protocols include <u>open shortest path first</u> (OSPF) and <u>intermediate system to intermediate system</u> (IS-IS).</p> <p>The link-state protocol is performed by every <i>switching node</i> in the network (i.e., nodes that are prepared to forward packets; in the <u>Internet</u>, these are called <u>routers</u>). The basic concept of link-state routing is that every node constructs a <i>map</i> of the connectivity to the network, in the form of a <u>graph</u>, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical <i>path</i> from it to every possible destination in the network. The collection of best paths will then form the node's <u>routing table</u>.</p> <p>This contrasts with <u>distance-vector routing protocols</u>, which work by having each node share its routing table with its neighbours. In a link-state protocol the only information passed between nodes is connectivity related.</p> <p>Link-state algorithms are sometimes characterized informally as each router 'telling the world about its neighbours'.</p> <pre> #include&lt;stdio.h&gt;  main() {     int n,a[10][10],i,j,k;      printf("\n ENTER THE NO.OF NODES: ");     scanf("%d",&amp;n);     printf("\n ENTER THE MATRIX ELEMENTS: ");     for(i=0;i&lt;n;i++)     {         printf("\nENTER THE DISTANCE FOR NODE:%d\n",i+1);         for(j=0;j&lt;n;j++)         { </pre>

```
scanf("%d",&a[i][j]);  
    }  
}  
for(i=0;i<n;i++)  
{  
    printf("THE LINK STATE STATE PACKETS FOR NODE:%d\n",i+1);  
    printf("\n NODE\tDISTANCE\n");  
    for(j=0;j<n;j++)  
    {  
        if(a[i][j]!=0&&a[i][j]!=-1)  
        {  
            printf("%d\t%d\n",j+1,a[i][j]);  
        }  
    }  
    printf("\n\n");  
}  
}
```

## Sample Output

```
"I:\Documents and Settings\diamond\Desktop\cn\link.exe"

ENTER THE NO.OF NODES: 3

ENTER THE MATRIX ELEMENTS:
ENTER THE DISTANCE FOR NODE:1
1 2 3

ENTER THE DISTANCE FOR NODE:2
2 3 4

ENTER THE DISTANCE FOR NODE:3
3 4 5
THE LINK STATE STATE PACKETS FOR NODE:1

  NODE    DISTANCE
  1        1
  2        2
  3        3

THE LINK STATE STATE PACKETS FOR NODE:2

  NODE    DISTANCE
  1        2
  2        3
  3        4

THE LINK STATE STATE PACKETS FOR NODE:3

  NODE    DISTANCE
  1        3
  2        4
  3        5

Press any key to continue . . . _
```



Experiment No:10	
Title	Implementation of Distance Vector algorithms.
Algorithms/Program	<p>In computer communication theory relating to packet-switched networks, a <b>distance-vector routing protocol</b> is one of the two major classes of routing protocols, the other major class being the link-state protocol. A distance-vector routing protocol uses the Bellman-Ford algorithm to calculate paths. A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically and, in some cases, when a change is detected in the topology of a network. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead. Distance Vector means that Routers are advertised as vector of distance and direction. 'Direction' is represented by next hop address and exit interface, whereas 'Distance' uses metrics such as hop count.</p> <p>Routers using distance vector protocol do not have knowledge of the entire path to a destination. Instead DV uses two methods:</p> <ol style="list-style-type: none"> <li>1. Direction in which or interface to which a packet should be forwarded.</li> <li>2. Distance from its destination.</li> </ol> <p>Examples of distance-vector routing protocols include Routing Information Protocol Version 1 &amp; 2, RIPv1 and RIPv2 and IGRP. EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.</p>
Algorithm/ Theory	<pre> #include&lt;stdio.h&gt;  struct node {     unsigned dist[20];     unsigned from[20]; }rt[10];  int main() {     int dmat[20][20];     int n,i,j,k,count=0;     printf("\nEnter the number of nodes : ");     scanf("%d",&amp;n);     printf("\nEnter the cost matrix :\n");     for(i=0;i&lt;n;i++)         for(j=0;j&lt;n;j++)         {             scanf("%d",&amp;dmat[i][j]);             dmat[i][i]=0;             rt[i].dist[j]=dmat[i][j];             rt[i].from[j]=j;         }     do </pre>

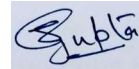
	<pre> {     count=0;     for(i=0;i&lt;n;i++)     for(j=0;j&lt;n;j++)     for(k=0;k&lt;n;k++)     if(rt[i].dist[j]&gt;dmat[i][k]+rt[k].dist[j])     {         rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];         rt[i].from[j]=k;         count++;     } }while(count!=0); for(i=0;i&lt;n;i++) {     printf("\n\nState value for router %d is \n",i+1);     for(j=0;j&lt;n;j++)     {         printf("\t\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);     } } printf("\n\n"); } </pre>
<b>Sample Output</b>	<p>Enter the number of nodes : 4</p> <p>Enter the cost matrix :</p> <pre> 0 3 5 99 3 0 99 1 5 4 0 2 99 1 2 0 </pre> <p>State value for router 1 is</p> <pre> node 1 via 1 Distance0 node 2 via 2 Distance3 node 3 via 3 Distance5 node 4 via 2 Distance4 </pre> <p>State value for router 2 is</p> <pre> node 1 via 1 Distance3 node 2 via 2 Distance0 node 3 via 4 Distance3 node 4 via 4 Distance1 </pre> <p>State value for router 3 is</p> <pre> node 1 via 1 Distance5 node 2 via 4 Distance3 node 3 via 3 Distance0 node 4 via 4 Distance2 </pre> <p>State value for router 4 is</p> <pre> node 1 via 2 Distance4 node 2 via 2 Distance1 node 3 via 3 Distance2 node 4 via 4 Distance0 </pre>

## **Value Added List of Experiments**

1. Take a 64-bit plain text and encrypt the same using DES algorithm
2. Implementation of Client-Server Communication Using TCP.
3. Simulate Sliding Window Protocol.
4. CLIENT – SERVER COMMUNICATION USING RPC.
5. Simulation of protocols ARP/RARP
6. Demonstration on NS-2

**SHILPY GUPTA**

**Name of the Course Coordinator:**



**Signature**