Hamming code

```
#include<stdlib.h>
#include<stdio.h>
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<4;i++)
printf("\t %s \n",gmatrix[i]);
printf("Encoded Data : ");
for(i=0;i<7;i++)
{
for(j=0;j<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<7;i++)
```

```
scanf("%d",&edata[i]);
for(i=0;i<3;i++){
for(j=0;j<7;j++)
syndrome[i]=syndrome[i]+(edata[j]*hmatrix[i][j]);
syndrome[i]=syndrome[i]%2;}
for(j=0;j<7;j++)
if
((syndrome[0]==hmatrix[0][j])&&(syndrome[1]==hmatrix[1][j
])&&
(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);
edata[j]=!edata[j];
printf("The Correct data Should be : ");
for(i=0;i<7;i++) printf(" %d ",edata[i]);
}}
```

CRC

```
// Online C compiler to run C program online
#include <stdio.h>
#include <string.h>
int main() {
   int i,j,keylen,msglen;
   char input[100],
key[30],temp[30],quot[100],rem[30],key1[30];
   printf("Enter Data: ");
   gets(input);
   printf("Enter Key: ");
   gets(key);
   keylen=strlen(key);
   msglen=strlen(input);
   strcpy(key1,key);
   for (i=0;i<keylen-1;i++) {
       input[msglen+i]='0';
   }
   for (i=0;i<keylen;i++)
    temp[i]=input[i];
   for (i=0;i<msglen;i++) {
       quot[i]=temp[0];
       if(quot[i]=='0')
        for (j=0;j<keylen;j++)
        key[j]='0'; else
        for (j=0;j<keylen;j++)
        key[j]=key1[j];
       for (j=keylen-1;j>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<msglen;i++)
     printf("%c",quot[i]);
    printf("\nRemainder is ");
    for (i=0;i<keylen-1;i++)
     printf("%c",rem[i]);
    printf("\nFinal data is: ");
    for (i=0;i<msglen;i++)
     printf("%c",input[i]);
    for (i=0;i<keylen-1;i++)
     printf("%c",rem[i]);
    return 0;
}
```

Dijkstra algorithm

```
// Dijkstra's Algorithm in C

#include <stdio.h>
#define INFINITY 9999
#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
  int cost[MAX][MAX], distance[MAX], pred[MAX];
  int visited[MAX], count, mindistance, nextnode, i, j;

  // Creating cost matrix
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (Graph[i][j] == 0)
        cost[i][j] = INFINITY;
      else
        cost[i][j] = Graph[i][j];

  for (i = 0; i < n; i++) {
    distance[i] = cost[start][i];
    pred[i] = start;
    visited[i] = 0;
  }

  distance[start] = 0;
  visited[start] = 1;
  count = 1;

  while (count < n - 1) {
    mindistance = INFINITY;

    for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
        mindistance = distance[i];
        nextnode = i;
      }

    visited[nextnode] = 1;
    for (i = 0; i < n; i++)
      if (!visited[i])
```

```
      if (mindistance + cost[nextnode][i] < distance[i]) {
        distance[i] = mindistance + cost[nextnode][i];
        pred[i] = nextnode;
      }
    count++;
  }

  // Printing the distance
  for (i = 0; i < n; i++)
    if (i != start) {
      printf("\nDistance from source to %d: %d", i, distance[i]);
    }
}
int main() {
  int Graph[MAX][MAX], i, j, n, u;
  n = 7;

  Graph[0][0] = 0;
  Graph[0][1] = 0;
  Graph[0][2] = 1;
  Graph[0][3] = 2;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

  Graph[1][0] = 0;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 3;
  Graph[1][6] = 0;

  Graph[2][0] = 1;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 1;
  Graph[2][4] = 3;
  Graph[2][5] = 0;
  Graph[2][6] = 0;

  Graph[3][0] = 2;
  Graph[3][1] = 0;
  Graph[3][2] = 1;
```

```
  Graph[3][3] = 0;
  Graph[3][4] = 0;
  Graph[3][5] = 0;
  Graph[3][6] = 1;

  Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 3;
  Graph[4][3] = 0;
  Graph[4][4] = 0;
  Graph[4][5] = 2;
  Graph[4][6] = 0;

  Graph[5][0] = 0;
  Graph[5][1] = 3;
  Graph[5][2] = 0;
  Graph[5][3] = 0;
  Graph[5][4] = 2;
  Graph[5][5] = 0;
  Graph[5][6] = 1;

  Graph[6][0] = 0;
  Graph[6][1] = 0;
  Graph[6][2] = 0;
  Graph[6][3] = 1;
  Graph[6][4] = 0;
  Graph[6][5] = 1;
  Graph[6][6] = 0;

  u = 0;
  Dijkstra(Graph, n, u);

  return 0;
}
```

Leaky bucket

```c
#include<stdio.h>

int main(){
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and no of inputs: ");
    scanf("%d %d %d", &buck_size, &outgoing, &n);

    while (n != 0) {
        printf("Enter the incoming packet size : ");
        scanf("%d", &incoming);
        printf("Incoming packet size %d\n", incoming);
        if (incoming <= (buck_size - store)){
            store += incoming;
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
        } else {
            printf("Dropped %d no of packets\n", incoming - (buck_size - store));
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
            store = buck_size;
        }
        store = store - outgoing;
        printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
        n--;
    }
}
```