**EXPERIMENT 1**

**AIM :** Write a program to implement bit stuffing.

**THEORY:**

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame

and the beginning of the next frame.

Bit stuffing is the insertion of non-informative bits into data.

**Note:** Stuffed bits should not be confused with overhead bits. Overhead bits are non-data bits that are necessary for transmission.

**Applications of Bit Stuffing -**

1. Synchronize several channels before multiplexing
2. Rate-match two single channels to each other
3. Run length limited coding

Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e.,

not corrupted by transmission errors). It is merely a way to ensure that the transmission

starts and ends at the correct places.

**Disadvantages of Bit Stuffing -**

The code rate is unpredictable, It depends on the data being transmitted.

**Example of bit stuffing -**

Bit sequence: 110101111101011111101011111110 (without bit stuffing)

Bit sequence: 110101111100101111101010111110110 (with bit stuffing)

After 5 consecutive 1-bits, a 0-bit is stuffed. Stuffed bits are marked bold.

**Algorithm for Bit−Stuffing**

* Start
* Initialize the array for transmitted stream with the special bit pattern 0111 1110 which indicates the beginning of the frame.
* Get the bit stream to be transmitted in to the array.
* Check for five consecutive ones and if they occur, stuff a bit 0
* Display the data transmitted as it appears on the data line after appending 0111 1110 at the end
* For de−stuffing, copy the transmitted data to another array after detecting the stuffed bits
* Display the received bit stream
* Stop

**CODE:**

#include<iostream>

#include<vector>

using namespace std;

vector<int> bit\_stuffing(vector<int> frame){

if(frame.size() < 5){

return frame;

}

vector<int> ans;

int i,j;

i = 0;

j = i;

while(j<frame.size()){

if(frame[j] != 1){

ans.push\_back(frame[j]);

i++;

j++;

}

else{

j = j + 1;

ans.push\_back(frame[i]);

while(j < frame.size() && frame[j] == 1){

ans.push\_back(frame[j]);

if(j - i + 1 == 5){

ans.push\_back(0);

}

j++;

}

if(j<frame.size())

ans.push\_back(frame[j]);

i = j + 1;

j = i;

}

}

return ans;

}

int main(){

int n;

cout<<"Enter the Number of bits in the frame : ";

cin>>n;

cout<<"Frame Before bit Stuffing : "<<endl;

vector<int> frame(n, 0);

for(int i = 0 ; i < frame.size() ; i++){

cin>>frame[i];

}

vector<int> ans = bit\_stuffing(frame);

cout<<"Frame After Bit Stuffing : "<<endl;

for(int i = 0 ; i < ans.size() ; i++){

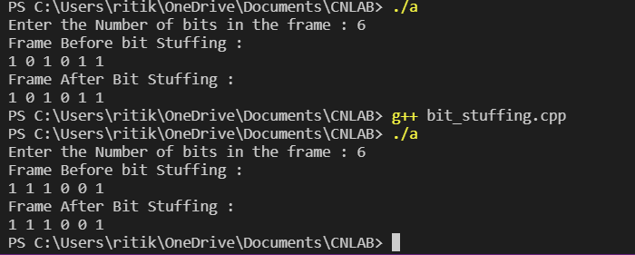
cout<<ans[i]<<" ";

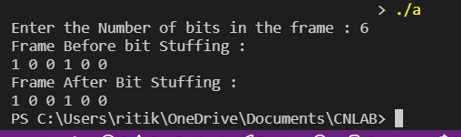
}

cout<<endl;

}

**OUTPUT :**





**CONCLUSION :**

Concept of bit stuffing is understood clearly, with all the disadvantages and applications.

The program was implemented correctly, passed all test and edge cases. Time complexity

of the program is О(N) and Space complexity is О(N).