NIHA
1BM18CS060

~~NAME : 1BM1~~

LAB-8

→ Write a program for error detecting using CRC (16 bit)

```
#include <stdio.h>
# define gen   0x11021
long int msb ( log int temp);
long int checksum (long int frame);

int main()
{
  long int opframe, inframe, flag, r_frame;
  printf ("Enter the frame");
  scanf ( "%lx", & inframe);
  inframe = inframe << 16;
  opframe = inframe ^ (checksum (inframe));
  printf ("The frame to be transmitted is
          %lx \n", opframe);
  printf ( "Enter the received frame : " );
  scanf ("%lx", & r_frame);
  printf ("For the received frame");
   flag = checksum( r_frame);
   if (flag > 0)
          printf ("\n Error !!! \n");
   else
       prinf ("\n data received is error free");

}
```

```
long int Checksum (long int frame)
{
    long int posit_frame, posit_gen, g=gen,
            g', temp;
   posit_frame = msb(frame);
  posit_gen = msb(gen);

   while(posit_gen <= posit_frame)
   {
        g1 = g<<(posit_frame - posit_gen);
        frame = g1 ^ frame;
        posit_frame = msb(frame);
  }
  printf("The checksum is %lx\n", frame);
   return (frame);
}

long int msb(long int temp)
{
      int i=0;
      if (temp == 0)
              return 0;
      while (temp >0)
      {
              temp = temp << 1;
              i++;
      }
}
```

⇒ write Dijkstra's algo to compute the shortest path for a given topology.

```
//pseudocode.

def dijsktra(graph, initial, end]:

    shortest_paths = {initial : (None, 0)}
    current_node = initial
    visited = set()
    ~~current_node = [illegible]~~
    while current_node != end :
        visited.add('current_node')
        destinations = graph.edges[current_node]
        weight_to_current_node =
                 shortest_paths[current_node][1]

      for next_node in destinations:
          weight = graph.weights[
           (current_node, next_node)] +
                  weight_to_current_node
          if next_node not in shortest_paths:
              shortest_paths[next_node] =
                   (current_node, weight)
          else:
              current_shortest_weight =
                   short_paths[next_node][1]
              if current_shortest_weight> weight
                  shortest_paths[next_node] =
                       (current_node, weight)
```

```
next_destination = { node : shortest_paths [node]
    for node in shortest_paths if node
        not in visited }

if not next_destinations :
        return "Route not possible"

current_node = min(next_destinations,
                key = lamda k: next_destination
                                    [k][1])

path = [ ]
while current_node is not None :
        path.append(current_nod)

        next_node = shortest_paths [current_node][0]

path = path[:: -1]
print('Shortest Weight :', current_shortest_weight)
print(path)
print('\n')
```