NAME OF THE EXPERIMENT: 3a) Shortest Path routing protocol

OBJECTIVE: Implement the data link layer framing method.

RESOURCE: Code blocks

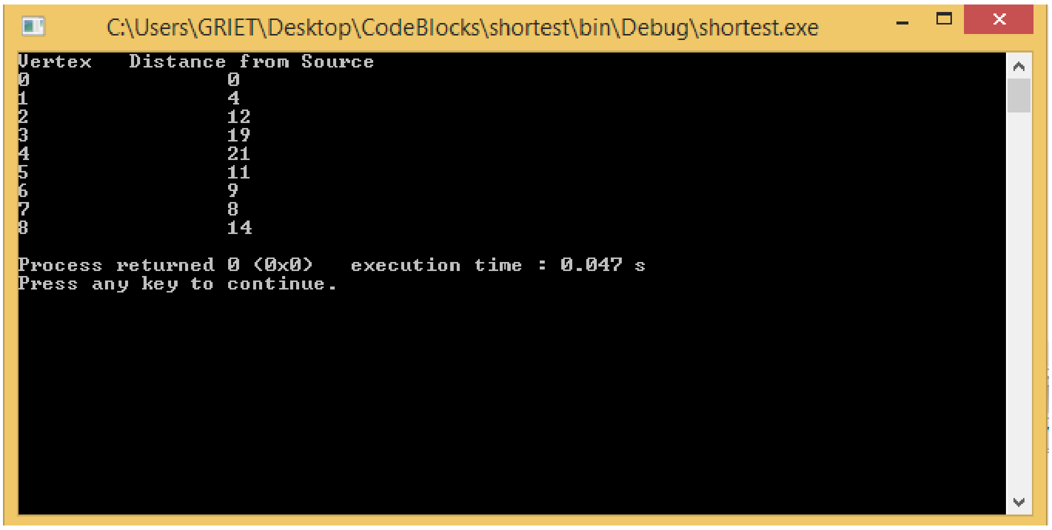
PROGRAM LOGIC: Dijkstra shortest path (SP): This algorithm finds the shortest route from a given source to a destination in a graph. The route is a path whose cost is the least possible one. K-shortest path (K-SP): K-shortest-path algorithms find more than one route for each source and destination pair.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#define V 9
int minDistance(int dist[], int sptSet[])
{
   int min = INT_MAX, min_index;
   int v;
   for (v = 0; v < V; v++)
      if (sptSet[v] == 0 && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}
void printSolution(int dist[], int n) {
   printf("Vertex   Distance from Source\n");
   int i;
   for (i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src) {
   int dist[V];
   int sptSet[V];
   int i, count, v;
   for (i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = 0;
   dist[src] = 0;
   for (count = 0; count < V - 1; count++) {
      int u = minDistance(dist, sptSet);
      sptSet[u] = 1;
      for (v = 0; v < V; v++)
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]
               + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }
   printSolution(dist, V);
}
int main() {
   /* Let us create the example graph discussed above */
```

```
   int graph[V][V] =  {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                      {4, 0, 8, 0, 0, 0, 0, 11, 0},
                      {0, 8, 0, 7, 0, 4, 0, 0, 2},
                      {0, 0, 7, 0, 9, 14, 0, 0, 0},
                      {0, 0, 0, 9, 0, 10, 0, 0, 0},
                      {0, 0, 4, 0, 10, 0, 2, 0, 0},
                      {0, 0, 0, 14, 0, 2, 0, 1, 6},
                      {8, 11, 0, 0, 0, 0, 1, 0, 7},
                      {0, 0, 2, 0, 0, 0, 6, 7, 0}
                     };
   dijkstra(graph, 0);
   return 0;
}
```

Output:



NAME OF THE EXPERIMENT: 3b)Distance Vector routing protocol

OBJECTIVE: Implement the data link layer framing method.

RESOURCE: Code blocks

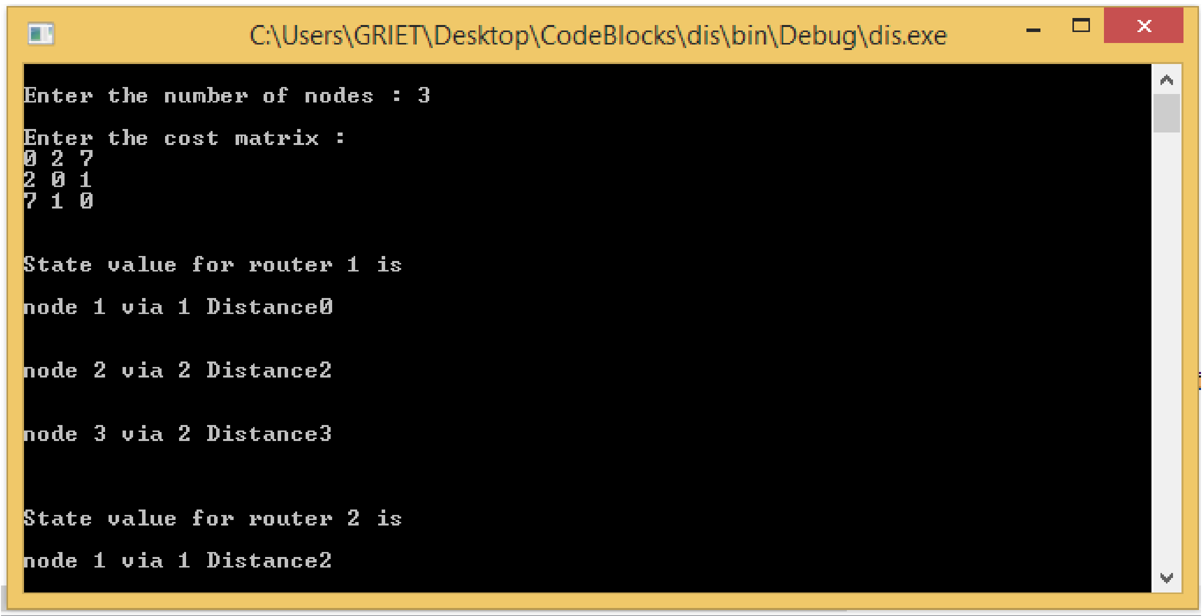
PROGRAM LOGIC: The distance vector routing algorithm is one of the most commonly used routing algorithms. It is a distributed algorithm, meaning that it is run on each router in the network. The algorithm works by each router sending updates to its neighbours about the best path to each destination.

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node  {
          unsigned dist[20];
```

```
        unsigned from[20];
}rt[10];
int main()  {
        int dmat[20][20],n,i,j,k,count=0;
        printf("\nEnter the number of nodes : ");
        scanf("%d",&n);
        printf("\nEnter the cost matrix :\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++ )  {
                        scanf("%d",&dmat[i][j]);
                        dmat[i][i]=0;
                        rt[i].dist[j]=dmat[i][j];
                        rt[i].from[j]=j;  }
                         do  {  count=0;
                        for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                        for(k=0;k<n;k++)
                                if(rt[i].dist[j]>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<n;i++)  {
                        printf("\n\nState value for router %d is \n",i+1);
                        for(j=0;j<n;j++)  {
                                printf("\t\nnode %d via %d
Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
        printf("\n\n");
                }
}
}
```

Output:



NAME OF THE EXPERIMENT: 3.C)Token Bucket algorithm

OBJECTIVE: Implement the data link layer framing method.

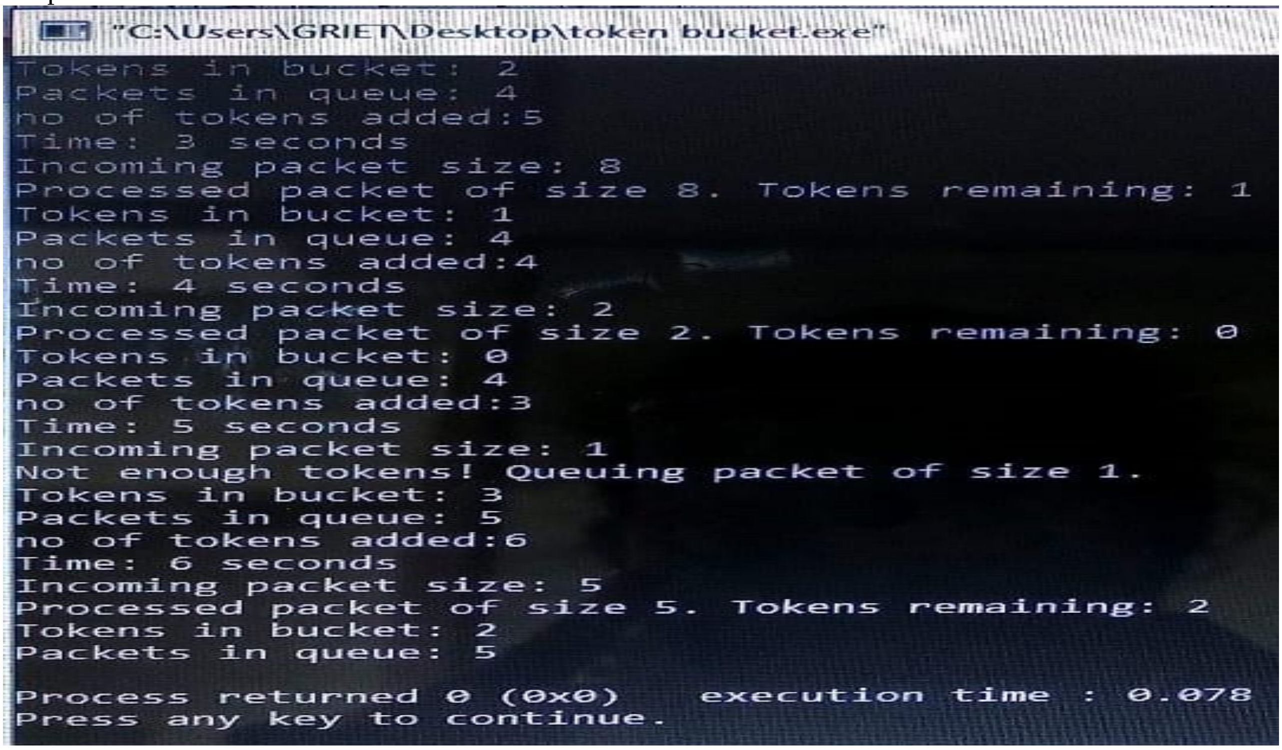
RESOURCE: Code blocks

PROGRAM LOGIC: Token bucket algorithm is one of the techniques for congestion control algorithms. When too many packets are present in the network it causes packet delay and loss of packet which degrades the performance of the system.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int bucket_size = 10;          // The maximum size of the bucket (token capacity)
    int tokens = 0;                // Current number of tokens in the bucket
    int token_rate = 3;            // Rate at which tokens are added (tokens per second)
    int no_of_queries = 6;         // Number of incoming packets (queries)
    int input_pkt_size[] = {4, 6, 8, 2, 1, 5};  // Incoming packet sizes
    int required_tokens = 4;       // Tokens required to process each packet
    int queue = 0;                 // Queue size for holding packets if tokens are insufficient
    int time_interval = 1;         // Time interval for adding tokens (simulated as 1 second)

    for (int i = 0; i < no_of_queries; i++) {
        // Add tokens at a constant rate
        tokens += token_rate * time_interval;
        printf("no of tokens added:%d",tokens);
        if (tokens > bucket_size) {
            tokens = bucket_size;  // Ensure tokens do not exceed bucket capacity
        }

        printf("\nTime: %d seconds\n", i+1);
        printf("Incoming packet size: %d\n", input_pkt_size[i]);

        // If not enough tokens for current packet, queue it
        if (tokens < required_tokens) {
            printf("Not enough tokens! Queuing packet of size %d.\n", input_pkt_size[i]);
            queue += input_pkt_size[i];  // Add to the queue
        }
        else {
            // Process the packet if enough tokens are available
            tokens -= required_tokens;
            printf("Processed packet of size %d. Tokens remaining: %d\n", input_pkt_size[i], tokens);

            // If there is a queue, process the queued packets when tokens become available
            while (queue > 0 && tokens >= required_tokens) {
                printf("Processing queued packet of size %d\n", queue);
                tokens -= required_tokens;
                queue = 0;  // Clear the queue after processing
            }
        }

        // Print current token and queue status
        printf("Tokens in bucket: %d\n", tokens);
        printf("Packets in queue: %d\n", queue);
    }
```

```
    return 0;
}
```
Output:



```
"C:\Users\GRIET\Desktop\token bucket.exe"
Tokens in bucket: 2
Packets in queue: 4
no of tokens added:5
Time: 3 seconds
Incoming packet size: 8
Processed packet of size 8. Tokens remaining: 1
Tokens in bucket: 1
Packets in queue: 4
no of tokens added:4
Time: 4 seconds
Incoming packet size: 2
Processed packet of size 2. Tokens remaining: 0
Tokens in bucket: 0
Packets in queue: 4
no of tokens added:3
Time: 5 seconds
Incoming packet size: 1
Not enough tokens! Queuing packet of size 1.
Tokens in bucket: 3
Packets in queue: 5
no of tokens added:6
Time: 6 seconds
Incoming packet size: 5
Processed packet of size 5. Tokens remaining: 2
Tokens in bucket: 2
Packets in queue: 5

Process returned 0 (0x0)   execution time : 0.078
Press any key to continue.
```