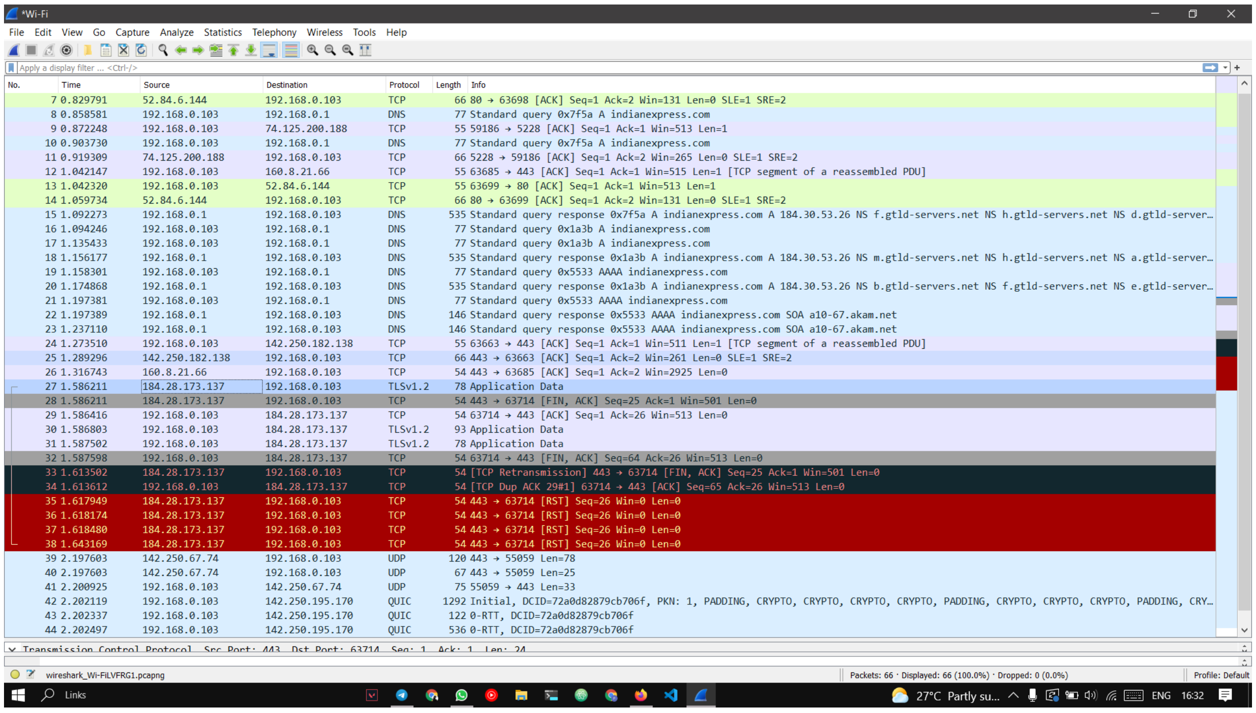
Name : P.Damodar.

Roll No : 411964.

CN-Week-4.

1. Perform packet sniffing using Wireshark and mention your observations with the Screenshots.

I started Capturing network and logged on to www.youtube.com My capturings :-

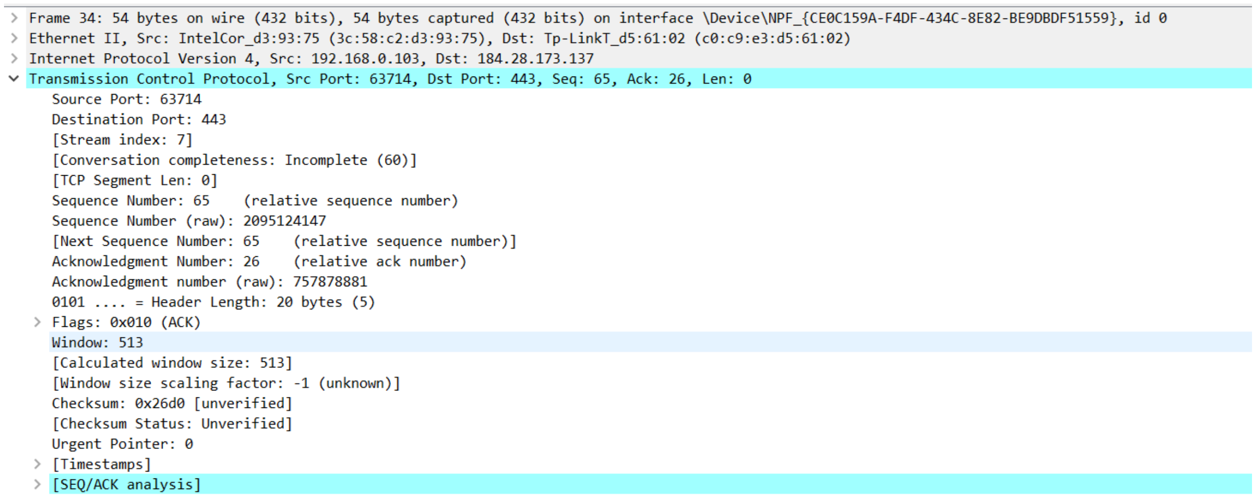

*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source          Destination     Protocol Length Info
  7 0.829791  52.84.6.144     192.168.0.103   TCP        66 80 → 63698 [ACK] Seq=1 Ack=2 Win=131 Len=0 SLE=1 SRE=2
  8 0.858581  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x7f5a A indianexpress.com
  9 0.872248  192.168.0.103   74.125.200.188  TCP        55 59186 → 5228 [ACK] Seq=1 Ack=1 Win=513 Len=1
 10 0.903730  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x7f5a A indianexpress.com
 11 0.919309  74.125.200.188  192.168.0.103   TCP        66 5228 → 59186 [ACK] Seq=1 Ack=2 Win=265 Len=0 SLE=1 SRE=2
 12 1.042147  192.168.0.103   160.8.21.66     TCP        55 63685 → 443 [ACK] Seq=1 Ack=1 Win=515 Len=1 [TCP segment of a reassembled PDU]
 13 1.042320  192.168.0.103   52.84.6.144     TCP        55 63699 → 80 [ACK] Seq=1 Ack=1 Win=513 Len=1
 14 1.059734  52.84.6.144     192.168.0.103   TCP        66 80 → 63699 [ACK] Seq=1 Ack=2 Win=131 Len=0 SLE=1 SRE=2
 15 1.092273  192.168.0.1     192.168.0.103   DNS       535 Standard query response 0x7f5a A indianexpress.com A 184.30.53.26 NS f.gtld-servers.net NS h.gtld-servers.net NS d.gtld-server…
 16 1.094246  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x1a3b A indianexpress.com
 17 1.135433  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x1a3b A indianexpress.com
 18 1.156177  192.168.0.1     192.168.0.103   DNS       535 Standard query response 0x1a3b A indianexpress.com A 184.30.53.26 NS m.gtld-servers.net NS h.gtld-servers.net NS a.gtld-server…
 19 1.158301  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x5533 AAAA indianexpress.com
 20 1.174868  192.168.0.1     192.168.0.103   DNS       535 Standard query response 0x1a3b A indianexpress.com A 184.30.53.26 NS b.gtld-servers.net NS f.gtld-servers.net NS e.gtld-server…
 21 1.197381  192.168.0.103   192.168.0.1     DNS        77 Standard query 0x5533 AAAA indianexpress.com
 22 1.197389  192.168.0.1     192.168.0.103   DNS       146 Standard query response 0x5533 AAAA indianexpress.com SOA a10-67.akam.net
 23 1.237110  192.168.0.1     192.168.0.103   DNS       146 Standard query response 0x5533 AAAA indianexpress.com SOA a10-67.akam.net
 24 1.273510  192.168.0.103   142.250.182.138 TCP        55 63663 → 443 [ACK] Seq=1 Ack=1 Win=511 Len=1 [TCP segment of a reassembled PDU]
 25 1.289296  142.250.182.138 192.168.0.103   TCP        66 443 → 63663 [ACK] Seq=1 Ack=2 Win=261 Len=0 SLE=1 SRE=2
 26 1.316743  160.8.21.66     192.168.0.103   TCP        54 443 → 63685 [ACK] Seq=1 Ack=2 Win=2925 Len=0
 27 1.586211  184.28.173.137  192.168.0.103   TLSv1.2    78 Application Data
 28 1.586211  184.28.173.137  192.168.0.103   TCP        54 443 → 63714 [FIN, ACK] Seq=25 Ack=1 Win=501 Len=0
 29 1.586416  192.168.0.103   184.28.173.137  TCP        54 63714 → 443 [ACK] Seq=1 Ack=26 Win=513 Len=0
 30 1.586803  192.168.0.103   184.28.173.137  TLSv1.2    93 Application Data
 31 1.587502  192.168.0.103   184.28.173.137  TLSv1.2    78 Application Data
 32 1.587598  192.168.0.103   184.28.173.137  TCP        54 63714 → 443 [FIN, ACK] Seq=64 Ack=26 Win=513 Len=0
 33 1.613502  184.28.173.137  192.168.0.103   TCP        54 [TCP Retransmission] 443 → 63714 [FIN, ACK] Seq=25 Ack=1 Win=501 Len=0
 34 1.613612  192.168.0.103   184.28.173.137  TCP        54 [TCP Dup ACK 29#1] 63714 → 443 [ACK] Seq=65 Ack=26 Win=513 Len=0
 35 1.617949  184.28.173.137  192.168.0.103   TCP        54 443 → 63714 [RST] Seq=26 Win=0 Len=0
 36 1.618174  184.28.173.137  192.168.0.103   TCP        54 443 → 63714 [RST] Seq=26 Win=0 Len=0
 37 1.618480  184.28.173.137  192.168.0.103   TCP        54 443 → 63714 [RST] Seq=26 Win=0 Len=0
 38 1.643169  184.28.173.137  192.168.0.103   TCP        54 443 → 63714 [RST] Seq=26 Win=0 Len=0
 39 2.197603  142.250.67.74   192.168.0.103   UDP       120 443 → 55059 Len=78
 40 2.197603  142.250.67.74   192.168.0.103   UDP        67 443 → 55059 Len=25
 41 2.200925  192.168.0.103   142.250.67.74   UDP        75 55059 → 443 Len=33
 42 2.202119  192.168.0.103   142.250.195.170 QUIC     1292 Initial, DCID=72a0d82879cb706f, PKN: 1, PADDING, CRYPTO, CRYPTO, CRYPTO, CRYPTO, PADDING, CRYPTO, CRYPTO, CRYPTO, PADDING, CRY…
 43 2.202337  192.168.0.103   142.250.195.170 QUIC      122 0-RTT, DCID=72a0d82879cb706f
 44 2.202497  192.168.0.103   142.250.195.170 QUIC      536 0-RTT, DCID=72a0d82879cb706f

Transmission Control Protocol, Src Port: 443, Dst Port: 63714, Seq: 1, Ack: 1, Len: 24

wireshark_Wi-FiLVFRG1.pcapng    Packets: 66 · Displayed: 66 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Got every packet with

1. Packet number - [1,2,3,4………………]
2. Source - an ip address
3. Destination - an ip address
4. Protocol - TCP/DNS/HTTP………
5. Length of the packet - 500/23/72………
6. Information about the packet -
   23   1.237110   192.168.0.1   192.168.0.103   DNS   146   Standard query response 0x5533 AAAA indianexpress.com SOA a10-67.akam.net

Examining A packet :-



```
> Frame 34: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{CE0C159A-F4DF-434C-8E82-BE9DBDF51559}, id 0
> Ethernet II, Src: IntelCor_d3:93:75 (3c:58:c2:d3:93:75), Dst: Tp-LinkT_d5:61:02 (c0:c9:e3:d5:61:02)
> Internet Protocol Version 4, Src: 192.168.0.103, Dst: 184.28.173.137
v Transmission Control Protocol, Src Port: 63714, Dst Port: 443, Seq: 65, Ack: 26, Len: 0
    Source Port: 63714
    Destination Port: 443
    [Stream index: 7]
    [Conversation completeness: Incomplete (60)]
    [TCP Segment Len: 0]
    Sequence Number: 65    (relative sequence number)
    Sequence Number (raw): 2095124147
    [Next Sequence Number: 65    (relative sequence number)]
    Acknowledgment Number: 26    (relative ack number)
    Acknowledgment number (raw): 757878881
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 513
    [Calculated window size: 513]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x26d0 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
```

1. Its frame number 34 of size 54 bytes => 432 bits.
2. Ethernet II with src and destination
   Src = InterCor……
   Dest = ${My Router i.e TPLink}
3. Internet Protocol Version - 4
   Src ip and destination ip
   Header checksum
4. TCP transmission control Protocol
   Source port
   Destination Port

The Data in the Packet :-

```
0000   c0 c9 e3 d5 61 02 3c 58 c2 d3 93 75 08 00 45
00   ....a.<X...u..E.
0010   00 28 08 15 40 00 80 06 00 00 c0 a8 00 67 b8
1c   .(..@........g..
0020   ad 89 f8 e2 01 bb 7c e1 0e b3 2d 2c 50 61 50
10   ......|...-,PaP.
0030   02 01 26 d0 00 00
    ..&...
```

2.Find Minimum Spanning Tree(MST) and Shortest Path Tree(SPT) for the graphs.

Graph 1

(a,c,6)

(a,b,6)

(a,d,6)

(b,d,2)

(c,d,2)

Graph 2

(a,c,3)

(a,b,3)

(a,d,3)

(b,d,1)

(c,d,1)

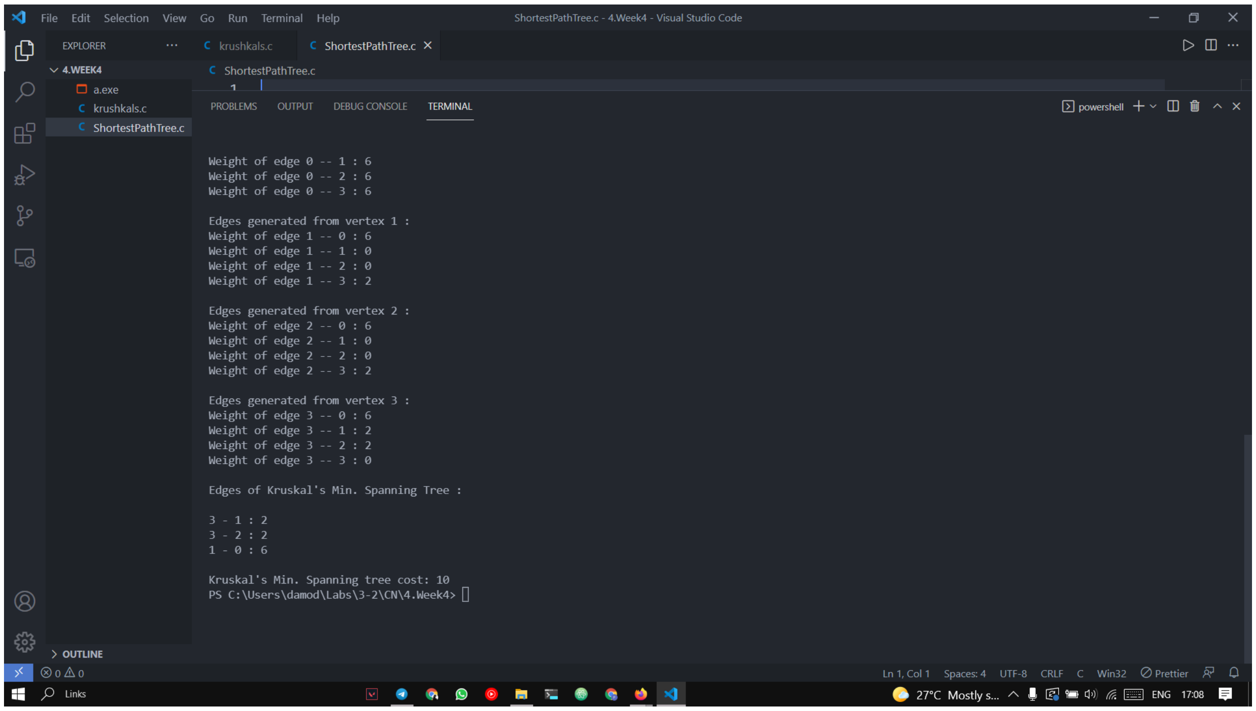
Minimum Spanning Tree :-

```
#include <stdio.h>

#define MAX 30

struct edge
{
    int u, v;    // Vertices connected by the edge
    int weight; // Weight of the connected edge
};
```

```
struct edgeList
{
    struct edge data[MAX];
    int n;
};

int G[MAX][MAX], n;

struct edgeList edgeSet;
struct edgeList span;

int find(int belongs[], int vno)
{
    return (belongs[vno]);
}

void applyUnion(int belongs[], int c1, int c2)

{
    int i;

    for (i = 0; i < n; i++)
    {
        if (belongs[i] == c2)
            belongs[i] = c1;
    }
}

void kruskal()
{
    int belongs[MAX], i = 0, j = 0, c1, c2;
    edgeSet.n = 0;

    // To generate the list of edges from the Adjacency
Matrix
    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
        {
            if (G[i][j] != 0)
            {
                edgeSet.data[edgeSet.n].u = i;
                edgeSet.data[edgeSet.n].v = j;
                edgeSet.data[edgeSet.n].weight = G[i][j];
                edgeSet.n++;
            }
        }

    // Sort the list of edges
    struct edge temp;

    for (i = 1; i < edgeSet.n; i++)
    {
        for (j = 0; j < edgeSet.n - 1; j++)
        {
            if (edgeSet.data[j].weight > edgeSet.data[j +
1].weight)
            {
                temp = edgeSet.data[j];
                edgeSet.data[j] = edgeSet.data[j + 1];
                edgeSet.data[j + 1] = temp;
            }
        }
    }
    for (i = 0; i < n; i++)
        belongs[i] = i;

    span.n = 0;

    for (i = 0; i < edgeSet.n; i++)
    {
        c1 = find(belongs, edgeSet.data[i].u);
        c2 = find(belongs, edgeSet.data[i].v);

        if (c1 != c2)
        {
            span.data[span.n] = edgeSet.data[i];
            span.n = span.n + 1;
            applyUnion(belongs, c1, c2);
        }
    }
}

void print()
{
    int i, cost = 0;

    printf("Edges of Kruskal's Min. Spanning Tree : \n");

    for (i = 0; i < span.n; i++)
    {
        printf("\n%d - %d : %d", span.data[i].u,
span.data[i].v, span.data[i].weight);
        cost = cost + span.data[i].weight;
    }
```

```c
    printf("\n\nKruskal's Min. Spanning tree cost: %d", cost);
}

int main()
    {
        printf("@@@@@@@@@@@@ KRUSKAL'S MINIMUM SPANNING TREE@@@@@@@@@@@@\n\n");

            printf("Enter the total no. of vertices in the graph : "); scanf("%d", &n);

            printf("\n");

            // Let the vertices of the Graph be from 0 to n-1

            /*Adjacency matrix for the Graph: a) For connected components -> Enter their weights
            b) For non-connected components -> Enter zero */

            printf("ENTER THE WEIGHTS ('0' for non-connected edges) :\n\n");

            int i = 0, j = 0;

            for (i = 0; i < n; i++)
            {
                printf("Edges generated from vertex %d : ", i);
                printf("\n");
                for (j = 0; j < n; j++)
                {
                    printf("Weight of edge %d -- %d : ",
i, j);

                    scanf("%d", &G[i][j]);
                }

                printf("\n");
            }

            kruskal(); print();

            return 0;
        }
```

Output :-



```
Weight of edge 0 -- 1 : 6
Weight of edge 0 -- 2 : 6
Weight of edge 0 -- 3 : 6

Edges generated from vertex 1 :
Weight of edge 1 -- 0 : 6
Weight of edge 1 -- 1 : 0
Weight of edge 1 -- 2 : 0
Weight of edge 1 -- 3 : 2

Edges generated from vertex 2 :
Weight of edge 2 -- 0 : 6
Weight of edge 2 -- 1 : 0
Weight of edge 2 -- 2 : 0
Weight of edge 2 -- 3 : 2

Edges generated from vertex 3 :
Weight of edge 3 -- 0 : 6
Weight of edge 3 -- 1 : 2
Weight of edge 3 -- 2 : 2
Weight of edge 3 -- 3 : 0

Edges of Kruskal's Min. Spanning Tree :

3 - 1 : 2
3 - 2 : 2
1 - 0 : 6

Kruskal's Min. Spanning tree cost: 10
PS C:\Users\damod\Labs\3-2\CN\4.Week4> 
```

Shortest Path Tree :-

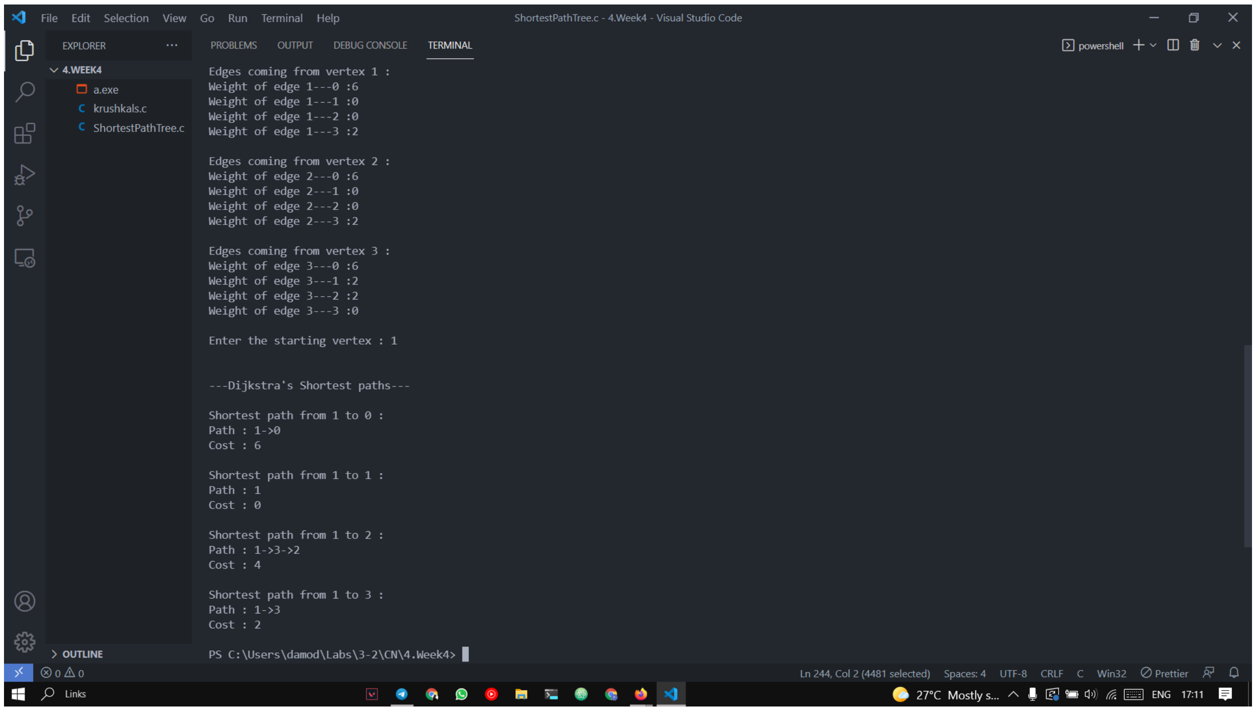
```c
#include <stdio.h>
#include <stdlib.h>
#define INFINITY 9999
#define MAX 10

struct node
{
    int data;
    int priority;
    struct node *next;
};

struct Stack
{
    int arr[MAX];
    int top;
};

void initStack(struct Stack *s)
{
    s->top = -1;
}

int isfull(struct Stack *s)
{
    if (s->top == MAX - 1)
        return 1;
    else
        return 0;
}

int isempty(struct Stack *s)
{
    if (s->top == -1)
        return 1;
    else
        return 0;
}

void push(struct Stack *s, int newitem)
{
    if (isfull(s))
    {
        printf("STACK FULL");
    }
    else
    {
        s->top++;
        s->arr[s->top] = newitem;
    }
}

int pop(struct Stack *s)
{
    int data;
    if (isempty(s))
    {
        printf("\n STACK EMPTY \n");
        return -1;
    }
    else
    {
        data = s->arr[s->top];
        s->top--;
    }

    return data;
}

// Create a new node
struct node *newNode(int d, int p)
{
    struct node *temp = (struct node
*)malloc(sizeof(struct node));
    temp->data = d;
    temp->priority = p;
    temp->next = NULL;

    return temp;
```

```
}

// To remove the element with highest priority
int extractMin(struct node **head)
{
    if (*head == NULL)

        return -1;

    int data = (*head)->data;
    struct node *temp = *head;
    (*head) = (*head)->next;
    free(temp);
    return data;
}

// To insert an element into priority queue

void enqueue(struct node **head, int d, int p)
{
    struct node *iter = (*head);
    struct node *temp = newNode(d, p);

    if (*head == NULL)
    {
        *head = temp;
        return;
    }

    if ((*head)->priority > p)
    {
        temp->next = *head;
        (*head) = temp;
    }
    else
    {
        while (iter->next != NULL && iter->next->priority
< p)
        {
            iter = iter->next;
        }

        temp->next = iter->next;
        iter->next = temp;
    }
}

void Dijkstra(int G[MAX][MAX], int n, int start)

{
    struct node *headPQ = NULL;

    int cost[MAX][MAX], i, j;

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

    int distance[MAX], parent[MAX];
    int S[MAX], u = 0, v = 0;

    distance[start] = 0;
    enqueue(&headPQ, start, 0);
    parent[start] = -1;
    S[start] = 0;

    for (i = 0; i < n; i++)
    {
        if (i != start)
        {
            distance[i] = INFINITY;
            parent[i] = -1;

            S[i] = 0;
        }
    }

    while (headPQ != NULL)
    {
        u = extractMin(&headPQ);
        S[u] = 1;

        for (v = 0; v < n; v++)
        {
            if (S[v] != 1)
            {
                if (distance[v] > distance[u] +
cost[u][v])
```

```
                {
                    distance[v] = distance[u] + cost[u][v];
                    enqueue(&headPQ, v, distance[v]);
                    parent[v] = u;
                }
            }
        }
    }

    struct Stack s;
    initStack(&s);

    printf("\n\n---Dijkstra's Shortest paths---\n\n");
    for (i = 0; i < n; i++)
    {
        printf("Shortest path from %d to %d :\n", start, i);
        printf("Path : ");
        int iter = i;
        while (parent[iter] != -1)
        {
            push(&s, iter);
            iter = parent[iter];
        }

        printf("%d", start);
        while (!isempty(&s))
        {
            printf("->%d", pop(&s));
        }
```

```c
        printf("\n");
        printf("Cost : %d\n\n", distance[i]);
    }
}

int main()
{
    int G[MAX][MAX];

    int n;
    int i = 0, j = 0;

    printf("@@@@@@@@@@ DIJKSTRA'S ALGORITHM
IMPLEMENTATION@@@@@@@@@@\n\n");
    printf("Enter the no. of vertices : ");

    scanf("%d", &n);
    printf("\n");

    printf("Enter the Adjacency Matrix for the Graph
:\n\n");

    for (i = 0; i < n; i++)
    {
        printf("Edges coming from vertex %d : \n", i);

        for (j = 0; j < n; j++)
        {
            printf("Weight of edge %d---%d :", i, j);

            scanf("%d", &G[i][j]);
        }
        printf("\n");
    }

    int start;
    printf("Enter the starting vertex : ");
    scanf("%d", &start);

    Dijkstra(G, n, start);

    return 0;
}
```

Output :-



```
Edges coming from vertex 1 :
Weight of edge 1---0 :6
Weight of edge 1---1 :0
Weight of edge 1---2 :0
Weight of edge 1---3 :2

Edges coming from vertex 2 :
Weight of edge 2---0 :6
Weight of edge 2---1 :0
Weight of edge 2---2 :0
Weight of edge 2---3 :2

Edges coming from vertex 3 :
Weight of edge 3---0 :6
Weight of edge 3---1 :2
Weight of edge 3---2 :2
Weight of edge 3---3 :0

Enter the starting vertex : 1


---Dijkstra's Shortest paths---

Shortest path from 1 to 0 :
Path : 1->0
Cost : 6

Shortest path from 1 to 1 :
Path : 1
Cost : 0

Shortest path from 1 to 2 :
Path : 1->3->2
Cost : 4

Shortest path from 1 to 3 :
Path : 1->3
Cost : 2

PS C:\Users\damod\Labs\3-2\CN\4.Week4> 
```

File Edit Selection View Go Run Terminal Help
ShortestPathTree.c - 4.Week4 - Visual Studio Code
EXPLORER
4.WEEK4
a.exe
krushkals.c
ShortestPathTree.c
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell
OUTLINE
Ln 244, Col 2 (4481 selected) Spaces: 4 UTF-8 CRLF C Win32 Prettier
Links
27°C Mostly s...
ENG 17:11

—---

-