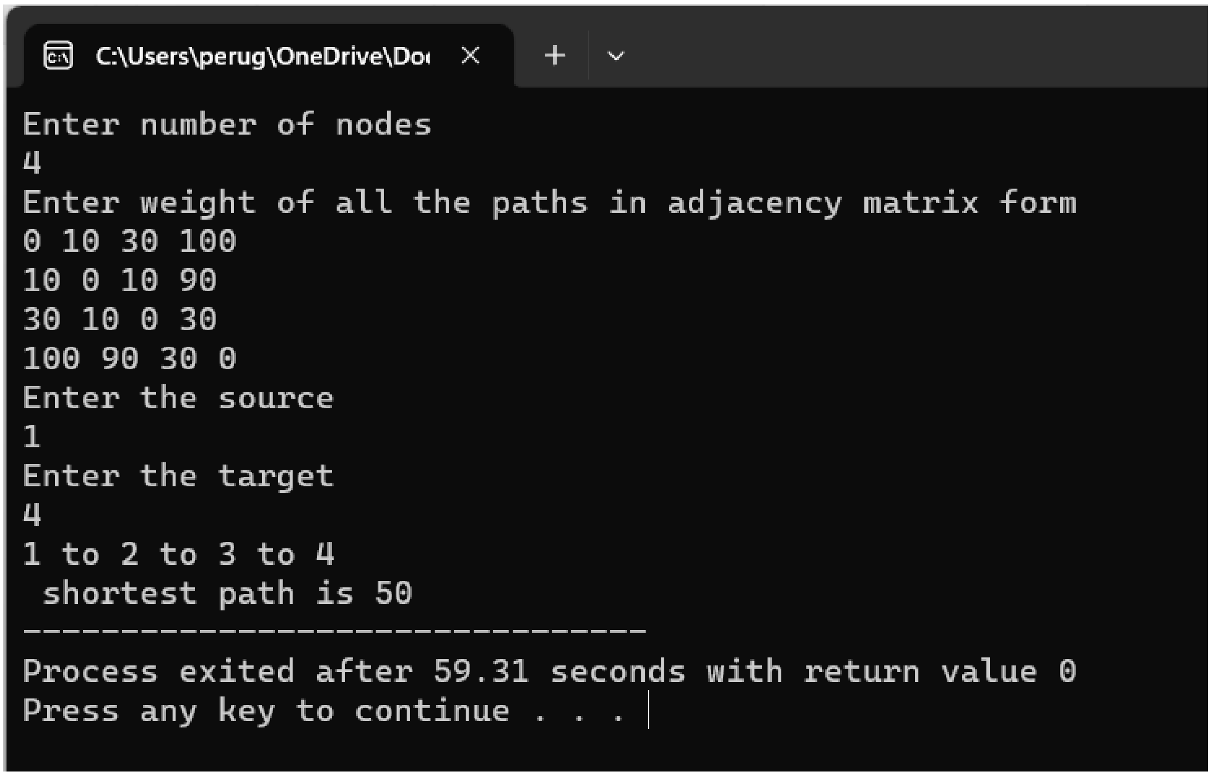
Data structure day -5

1.write c program to implement single source shortest path technique?

```
#include<stdio.h>
int main()
{
    int
cost[10][10],i,j,n,source,target,visited[10]={0},min=99
9,dist[10],pre[10];
    int start,m,d,path[10];
    printf("Enter number of nodes\n ");
    scanf("%d",&n);
    printf("Enter weight of all the paths in adjacency
matrix form\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
              cost[i][j]=999;
        }
    }
```

```
printf("Enter the source\n");
scanf("%d",&source);
printf("Enter the target\n");
scanf("%d",&target);
start=source;
for(i=1;i<=n;i++)
{
   dist[i]=999;
   pre[i]=-1;
}
visited[source]=1;
dist[source]=0;
while(visited[target]==0)
{
   min=999;
   m=0;
   for(i=1;i<=n;i++)
   {
      d=dist[start]+cost[start][i];
      if(d<dist[i] && visited[i]==0)
      {
         dist[i]=d;
         pre[i]=start;
      }
      if(min>dist[i] && visited[i]==0)
      {
```

```
                min=dist[i];
                m=i;
            }
        }
        start=m;
        visited[m]=1;
    }
    start=target;
    j=0;
    while(start!=-1)
    {
        path[j++]=start;
        start=pre[start];
    }
    for(i=j-1;i>=0;i--)
    {
        if(i!=j-1)
          printf(" to ");
         printf("%d",path[i]);
    }

    printf("\n shortest path is %d",dist[target]);
    return 0;
}
```



```
Enter number of nodes
4
Enter weight of all the paths in adjacency matrix form
0 10 30 100
10 0 10 90
30 10 0 30
100 90 30 0
Enter the source
1
Enter the target
4
1 to 2 to 3 to 4
 shortest path is 50
--------------------------------
Process exited after 59.31 seconds with return value 0
Press any key to continue . . .
```

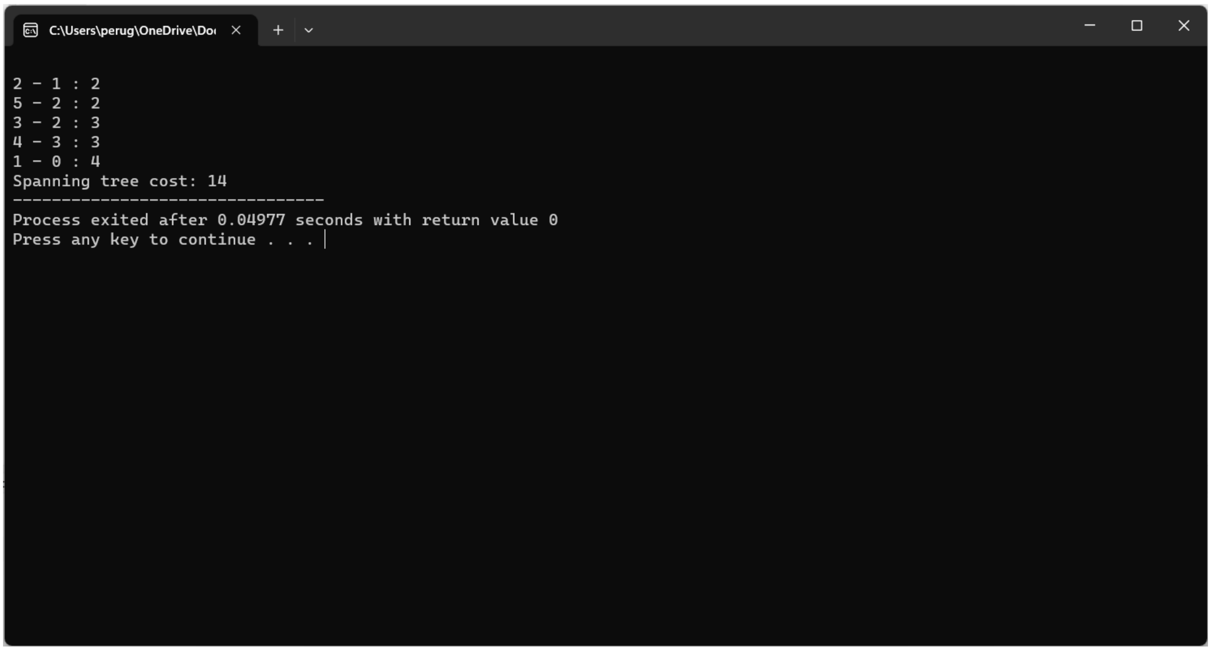
2.write c program to implement minimum spanning tree using kruskal’s algorithm ?

```
// Kruskal's algorithm in C

#include <stdio.h>

#define MAX 30

typedef struct edge {
  int u, v, w;
} edge;

typedef struct edge_list {
  edge data[MAX];
  int n;
```

```
} edge_list;

edge_list elist;

int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;

  for (i = 1; i < n; i++)
    for (j = 0; j < i; j++) {
      if (Graph[i][j] != 0) {
        elist.data[elist.n].u = i;
        elist.data[elist.n].v = j;
        elist.data[elist.n].w = Graph[i][j];
        elist.n++;
      }
    }

  sort();
  for (i = 0; i < n; i++)
    belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {
    cno1 = find(belongs, elist.data[i].u);
    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
      spanlist.data[spanlist.n] = elist.data[i];
      spanlist.n = spanlist.n + 1;
      applyUnion(belongs, cno1, cno2);
    }
  }
}

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

void applyUnion(int belongs[], int c1, int c2) {
  int i;

  for (i = 0; i < n; i++)
    if (belongs[i] == c2)
```

```
      belongs[i] = c1;
}
void sort() {
  int i, j;
  edge temp;

  for (i = 1; i < elist.n; i++)
    for (j = 0; j < elist.n - 1; j++)
      if (elist.data[j].w > elist.data[j + 1].w) {
        temp = elist.data[j];
        elist.data[j] = elist.data[j + 1];
        elist.data[j + 1] = temp;
      }
}
void print() {
  int i, cost = 0;

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

  printf("\nSpanning tree cost: %d", cost);
}

int main() {
  int i, j, total_cost;
    n = 6;
 Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;

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

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

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

  kruskalAlgo();
  print();
}
```

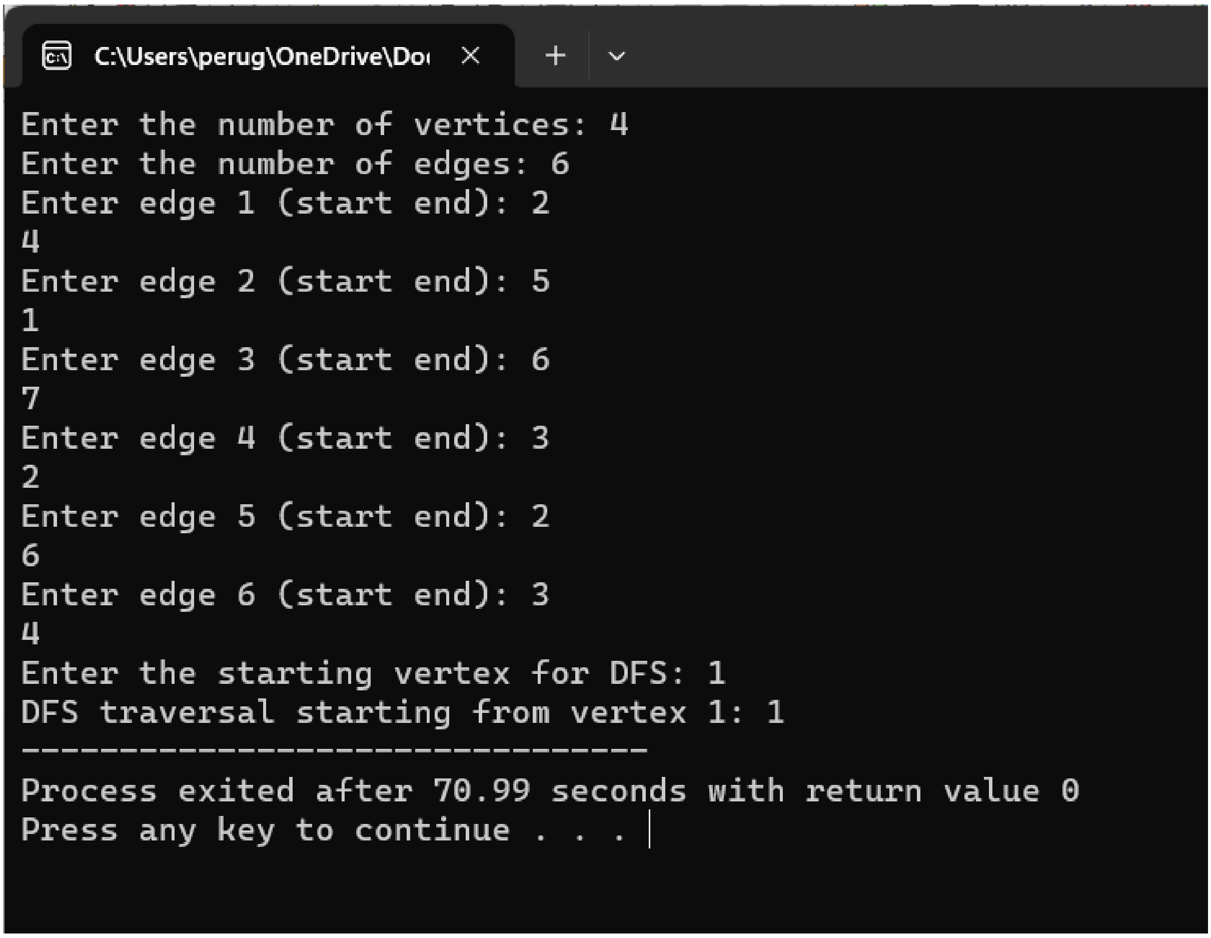


```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
--------------------------------
Process exited after 0.04977 seconds with return value 0
Press any key to continue . . .
```

3.write c program to implement depth for search graph traversal?

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
bool visited[MAX_VERTICES];
int
adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

void initialize() {
    for (int i = 0; i < MAX_VERTICES; i++) {
        visited[i] = false;
        for (int j = 0; j < MAX_VERTICES; j++) {
            adjacencyMatrix[i][j] = 0;
        }
    }
```

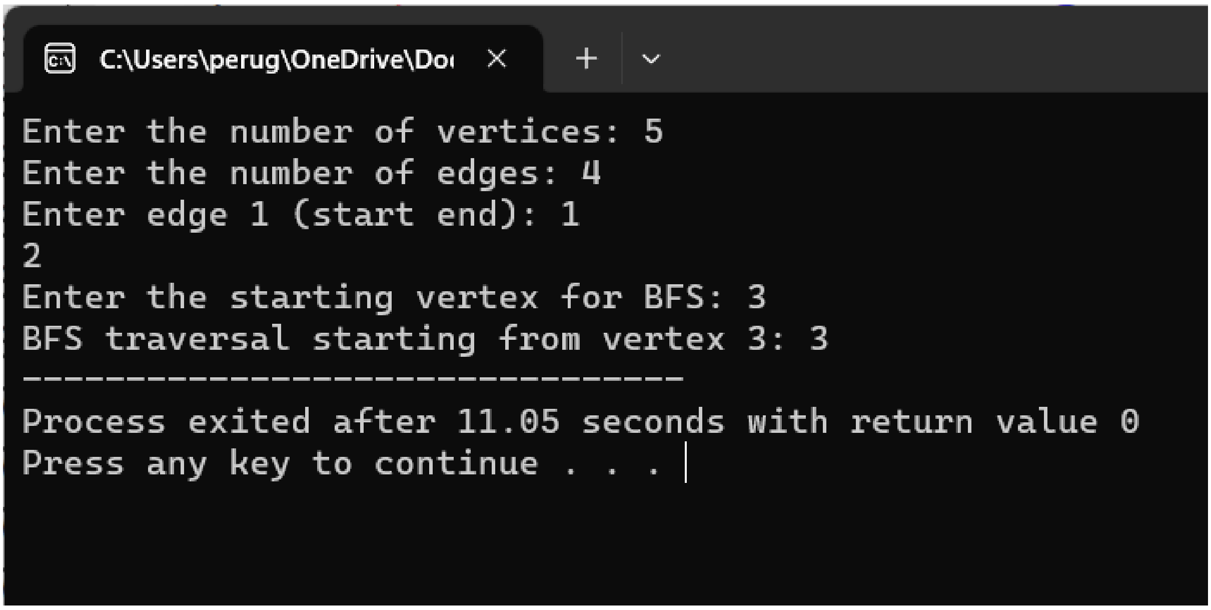
```
}

void addEdge(int start, int end) {
    adjacencyMatrix[start][end] = 1;
    adjacencyMatrix[end][start] = 1;
}

void DFS(int vertex) {
    visited[vertex] = true;
    printf("%d ", vertex);

    for (int i = 0; i < numVertices; i++) {
        if (adjacencyMatrix[vertex][i] && !visited[i]) {
            DFS(i);
        }
    }
}

int main() {
    initialise();
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numEdges; i++) {
        int start, end;
        printf("Enter edge %d (start end): ", i + 1);
        scanf("%d %d", &start, &end);
        addEdge(start, end);
    }
    int startVertex;
    printf("Enter the starting vertex for DFS: ");
    scanf("%d", &startVertex);

    printf("DFS traversal starting from vertex %d: ",
startVertex);
    DFS(startVertex);

    return 0;
}
```



```
Enter the number of vertices: 4
Enter the number of edges: 6
Enter edge 1 (start end): 2
4
Enter edge 2 (start end): 5
1
Enter edge 3 (start end): 6
7
Enter edge 4 (start end): 3
2
Enter edge 5 (start end): 2
6
Enter edge 6 (start end): 3
4
Enter the starting vertex for DFS: 1
DFS traversal starting from vertex 1: 1
--------------------------------
Process exited after 70.99 seconds with return value 0
Press any key to continue . . .
```

3.write c program to implement BFS graph traversal?

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
#define QUEUE_SIZE 100
bool visited[MAX_VERTICES];
int
adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;
int queue[QUEUE_SIZE];
int front = -1, rear = -1;
void initialize() {
    for (int i = 0; i < MAX_VERTICES; i++) {
        visited[i] = false;
        for (int j = 0; j < MAX_VERTICES; j++) {
            adjacencyMatrix[i][j] = 0;
        }
    }
}
void addEdge(int start, int end) {
    adjacencyMatrix[start][end] = 1;
    adjacencyMatrix[end][start] = 1;
}
void enqueue(int vertex) {
    if (rear == QUEUE_SIZE - 1) {
        printf("Queue is full.\n");
        return;
    }
    if (front == -1)
        front = 0;
    rear++;
    queue[rear] = vertex;
}
int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
        return -1;
    }
    int vertex = queue[front];
    front++;
    return vertex;
}

void BFS(int startVertex) {
    visited[startVertex] = true;
    enqueue(startVertex);
    while (front <= rear) {
        int currentVertex = dequeue();
        printf("%d ", currentVertex);

        for (int i = 0; i < numVertices; i++) {
            if (adjacencyMatrix[currentVertex][i] &&
!visited[i]) {
                visited[i] = true;
                enqueue(i);
            }
        }
    }
}
int main() {
    initialize();
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);
    for (int i = 0; i < numEdges; i++)
        {
        int start, end;
        printf("Enter edge %d (start end): ", i + 1);
        scanf("%d %d", &start, &end)
        addEdge(start, end);
    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);
    printf("BFS traversal starting from vertex %d: ",
startVertex);
    BFS(startVertex);
    return 0;
}
}
```



```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge 1 (start end): 1
2
Enter the starting vertex for BFS: 3
BFS traversal starting from vertex 3: 3
--------------------------------
Process exited after 11.05 seconds with return value 0
Press any key to continue . . .
```