WEEK –10

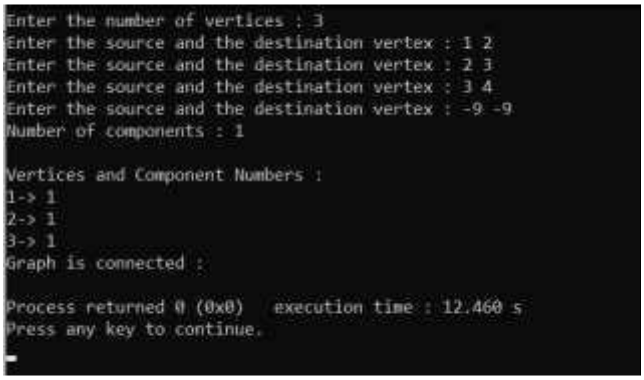
TOPIC : GRAPHS

1. Write a program to check the connectivity of a graph using DFS (Recursion). Also compute the number of components of a graph.

```
#include <stdio.h>
int visit[100]; void
create_graph(); void
DFS(int); int
a[100][100], n; int
component(); int
label; int connect();
int main()
{  int i, v,
k;
 printf("Enter the number of vertices : ");
 scanf("%d", &n);  create_graph();  int result =
component();  printf("Number of components :
%d\n", result);  printf("\nVertices and Component
Numbers : \n");  for (i = 1; i <= n; i++)
printf("%d-> %d\n", i, visit[i]);  int result2 =
connect();
 if (result2)  printf("Graph is
connected : \n");  else
 printf("Graph is not connected : \n");
}
int component()
{
 int i, j;
label = 0;
 for (i = 1; i <= n; i++)
 {
 if (visit[i] == 0)
 {
label++;
 DFS(i);
 }
 }  return
label; }
void create_graph()
```

```
{
 int i, j;
 while (1)
{
 printf("Enter the source and the destination vertex : ");
 scanf("%d%d", &i, &j);
if ((i == -9) && (j == -9))
break;  a[i][j] = a[j][i] = 1;
 } } void DFS(int v) {
int u;  visit[v] = label;
for (u = 1; u <= n; u++)
 {
 if ((a[v][u] == 1) && (visit[u] == 0))
 DFS(u);
 } } int
connect() {
 int i;  for (i = 1; i <=
n; i++)
 visit[i] = 0;  DFS(1);
for (i = 1; i <= n; i++)
 {  if (visit[i] ==
0)  return 0;  }
return 1;
}
```

OUTPUT:



```
Enter the number of vertices : 3
Enter the source and the destination vertex : 1 2
Enter the source and the destination vertex : 2 3
Enter the source and the destination vertex : 3 4
Enter the source and the destination vertex : -9 -9
Number of components : 1

Vertices and Component Numbers :
1-> 1
2-> 1
3-> 1
Graph is connected :

Process returned 0 (0x0)   execution time : 12.460 s
Press any key to continue.
```

2. **Write a program traverse the graph using BFS traversal technique. Use an explicit queue for implementation.**

```
// BFS algorithm in C

#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
int items[SIZE];
```

```
int front;   int
rear;
};

struct queue* createQueue(); void
enqueue(struct queue* q, int); int
dequeue(struct queue* q); void
display(struct queue* q); int
isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {   int
vertex;   struct
node* next;
};

struct node* createNode(int);

struct Graph {   int
numVertices;   struct
node** adjLists;
  int* visited;
};

// BFS algorithm void bfs(struct Graph* graph,
int startVertex) {   struct queue* q =
createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
printQueue(q);     int
currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {         graph-
>visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
```

```
}

// Creating a node struct node* createNode(int v) {
struct node* newNode = malloc(sizeof(struct node));
newNode->vertex = v;   newNode->next = NULL;
return newNode;
}

// Creating a graph struct Graph* createGraph(int
vertices) {   struct Graph* graph =
malloc(sizeof(struct Graph));   graph->numVertices
= vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {     graph-
>adjLists[i] = NULL;     graph->visited[i] = 0;
  }

  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest   struct node*
newNode = createNode(dest);   newNode-
>next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src   newNode
= createNode(src);   newNode->next =
graph->adjLists[dest];   graph-
>adjLists[dest] = newNode;
}

// Create a queue struct queue* createQueue() {
struct queue* q = malloc(sizeof(struct queue));
q->front = -1;   q->rear = -1;
  return q; }

// Check if the queue is empty
int isEmpty(struct queue* q) {
if (q->rear == -1)     return 1;
else     return 0;
}
```

```
// Adding elements into queue void
enqueue(struct queue* q, int value) {   if
(q->rear == SIZE - 1)     printf("\nQueue
is Full!!");   else {     if (q->front == -1)
q->front = 0;     q->rear++;
    q->items[q->rear] = value;
  }
}

// Removing elements from queue int
dequeue(struct queue* q) {
  int item;   if (isEmpty(q)) {
printf("Queue is empty");
item = -1;   } else {     item =
q->items[q->front];     q-
>front++;     if (q->front > q-
>rear) {       printf("Resetting
queue ");
      q->front = q->rear = -1;
    }   }
return item;
}

// Print the queue void
printQueue(struct queue* q) {   int
i = q->front;

  if (isEmpty(q)) {
printf("Queue is empty");
  } else {     printf("\nQueue contains \n");
for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
  struct Graph* graph = createGraph(6);
addEdge(graph, 0, 1);   addEdge(graph,
0, 2);   addEdge(graph, 1, 2);
addEdge(graph, 1, 4);   addEdge(graph,
1, 3);   addEdge(graph, 2, 4);
addEdge(graph, 3, 4);

  bfs(graph, 0);

  return 0;
}
```

OUTPUT:

```
Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3

Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```