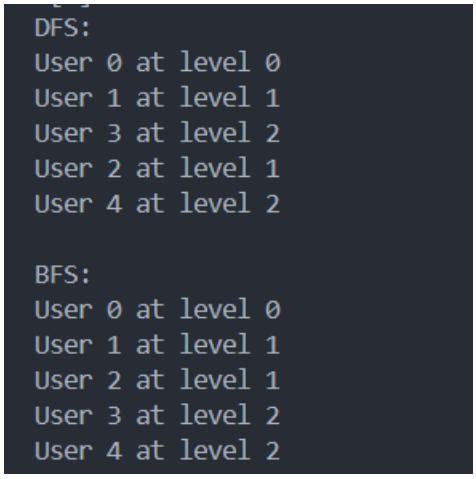
ASSIGNMENT – 8

Q1)

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

#define MAX 100

typedef struct {
    int adj[MAX][MAX];
    int visited[MAX];
    int n;
} Graph;

void initGraph(Graph *g, int n) {
    g->n = n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            g->adj[i][j] = 0;
        }
        g->visited[i] = 0;
    }
}

void addEdge(Graph *g, int u, int v) {
    g->adj[u][v] = 1;
    g->adj[v][u] = 1;
}

void DFS(Graph *g, int start, int level, int k) {
```

```
    if (level > k) return;
    g->visited[start] = 1;
    printf("User %d at level %d\n", start, level);
    for (int i = 0; i < g->n; i++) {
        if (g->adj[start][i] && !g->visited[i]) {
            DFS(g, i, level + 1, k);
        }
    }
}

void BFS(Graph *g, int start, int k) {
    int queue[MAX], front = 0, rear = 0, level[MAX] = {0};
    g->visited[start] = 1;
    queue[rear++] = start;
    while (front < rear) {
        int u = queue[front++];
        if (level[u] > k) break;
        printf("User %d at level %d\n", u, level[u]);
        for (int i = 0; i < g->n; i++) {
            if (g->adj[u][i] && !g->visited[i]) {
                g->visited[i] = 1;
                queue[rear++] = i;
                level[i] = level[u] + 1;
            }
        }
    }
}

void resetVisited(Graph *g) {
    for (int i = 0; i < g->n; i++) {
        g->visited[i] = 0;
```

```
    }
}

int main() {
    Graph g;
    int n = 6; // Number of users
    initGraph(&g, n);

    addEdge(&g, 0, 1);
    addEdge(&g, 0, 2);
    addEdge(&g, 1, 3);
    addEdge(&g, 2, 4);
    addEdge(&g, 3, 4);
    addEdge(&g, 4, 5);

    int startUser = 0;
    int k = 2;

    printf("DFS:\n");
    DFS(&g, startUser, 0, k);
    resetVisited(&g);

    printf("\nBFS:\n");
    BFS(&g, startUser, k);

    return 0;
}
```



BFS is preferred for finding nodes within kkk levels in an unweighted graph because it processes nodes by distance from the start. DFS is more efficient in deep, sparsely connected graphs where long paths need to be explored quickly.

Q2)

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_NODES 100


typedef struct Node

{

   int vertex;

   struct Node *next;

} Node;


typedef struct Graph

{

   int numVertices;

   Node **adjLists;

   int *visited;

} Graph;
```

```
typedef struct Queue
{
   int items[MAX_NODES];
   int front;
   int rear;
} Queue;

Queue *createQueue()
{
   Queue *q = (Queue *)malloc(sizeof(Queue));
   q->front = -1;
   q->rear = -1;
   return q;
}

int isEmpty(Queue *q)
{
   return q->rear == -1;
}

void enqueue(Queue *q, int value)
{
   if (q->rear == MAX_NODES - 1)
      return;
   if (isEmpty(q))
   {
      q->front = 0;
   }
   q->rear++;
   q->items[q->rear] = value;
}
```

```
int dequeue(Queue *q)
{
   int item;
   if (isEmpty(q))
   {
      return -1;
   }
   item = q->items[q->front];
   q->front++;
   if (q->front > q->rear)
   {
      q->front = q->rear = -1;
   }
   return item;
}

Graph *createGraph(int vertices)
{
   Graph *graph = (Graph *)malloc(sizeof(Graph));
   graph->numVertices = vertices;
   graph->adjLists = (Node **)malloc(vertices * sizeof(Node *));
   graph->visited = (int *)malloc(vertices * sizeof(int));
   for (int i = 0; i < vertices; i++)
   {
      graph->adjLists[i] = NULL;
      graph->visited[i] = 0;
   }
   return graph;
}
```

```
Node *createNode(int v)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

void addEdge(Graph *graph, int src, int dest)
{
    Node *newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void dfs(Graph *graph, int vertex, int targetVertex, int *path, int *pathIndex)
{
    graph->visited[vertex] = 1;
    path[(*pathIndex)++] = vertex;

    if (vertex == targetVertex)
    {
        printf("Path (DFS): ");
        for (int i = 0; i < *pathIndex; i++)
        {
            printf("%d ", path[i]);
        }
```

```
        printf("\n");
        return;
    }

    Node *temp = graph->adjLists[vertex];
    while (temp)
    {
        int adjVertex = temp->vertex;
        if (!graph->visited[adjVertex])
        {
            dfs(graph, adjVertex, targetVertex, path, pathIndex);
        }
        temp = temp->next;
    }

    (*pathIndex)--;
}

void resetVisited(Graph *graph)
{
    for (int i = 0; i < graph->numVertices; i++)
    {
        graph->visited[i] = 0;
    }
}

int main()
{
    int vertices = 13;
    Graph *graph = createGraph(vertices);
```

```
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 5);
    addEdge(graph, 2, 6);
    addEdge(graph, 3, 7);
    addEdge(graph, 3, 8);
    addEdge(graph, 4, 9);
    addEdge(graph, 6, 10);
    addEdge(graph, 7, 11);
    addEdge(graph, 9, 12);

    int startVertex = 0;
    int targetVertex = 5;

    resetVisited(graph);

    int path[MAX_NODES], pathIndex = 0;
    dfs(graph, startVertex, targetVertex, path, &pathIndex);

    return 0;
}
```

```
Path (DFS): 0 2 5
```