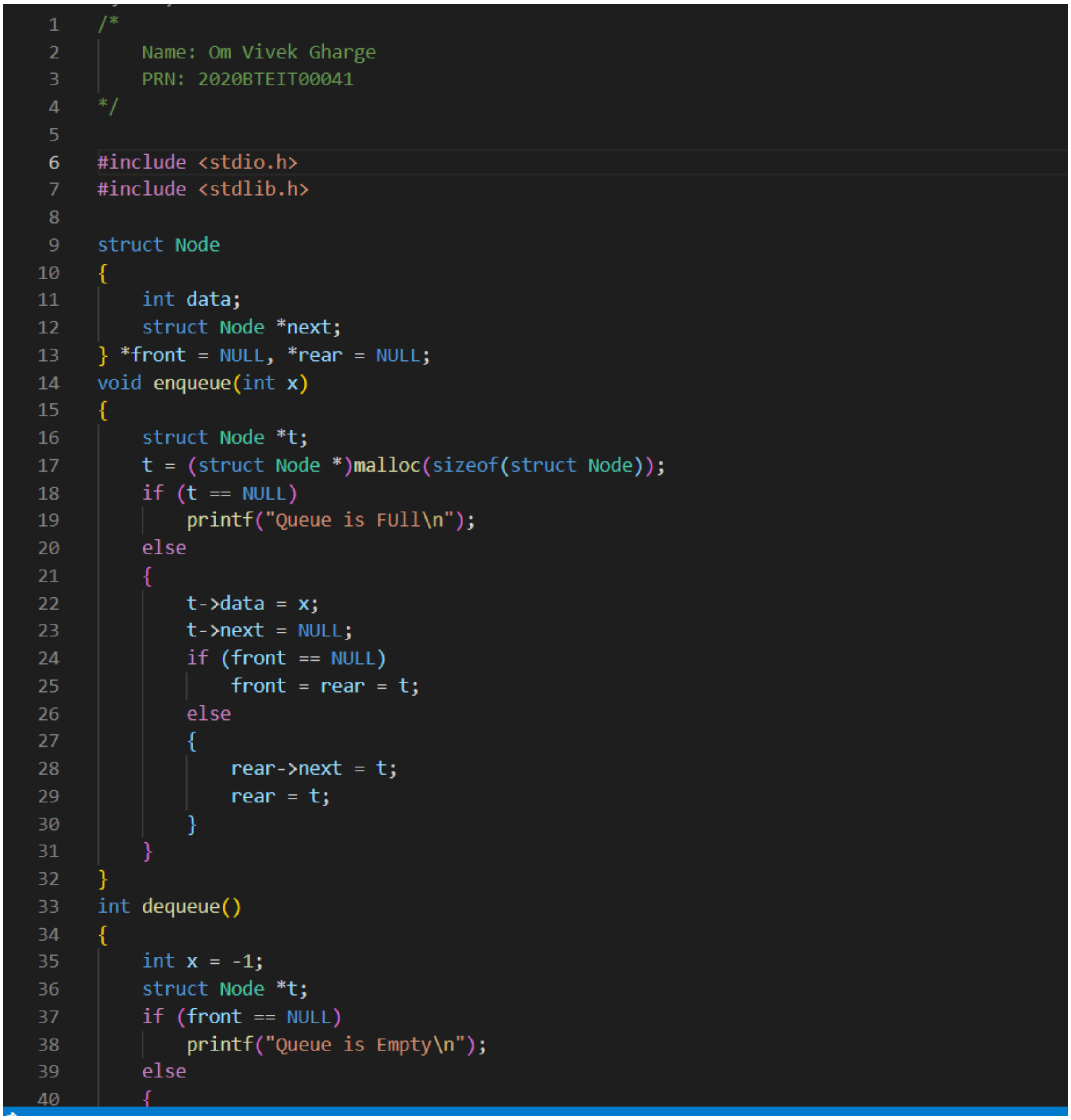
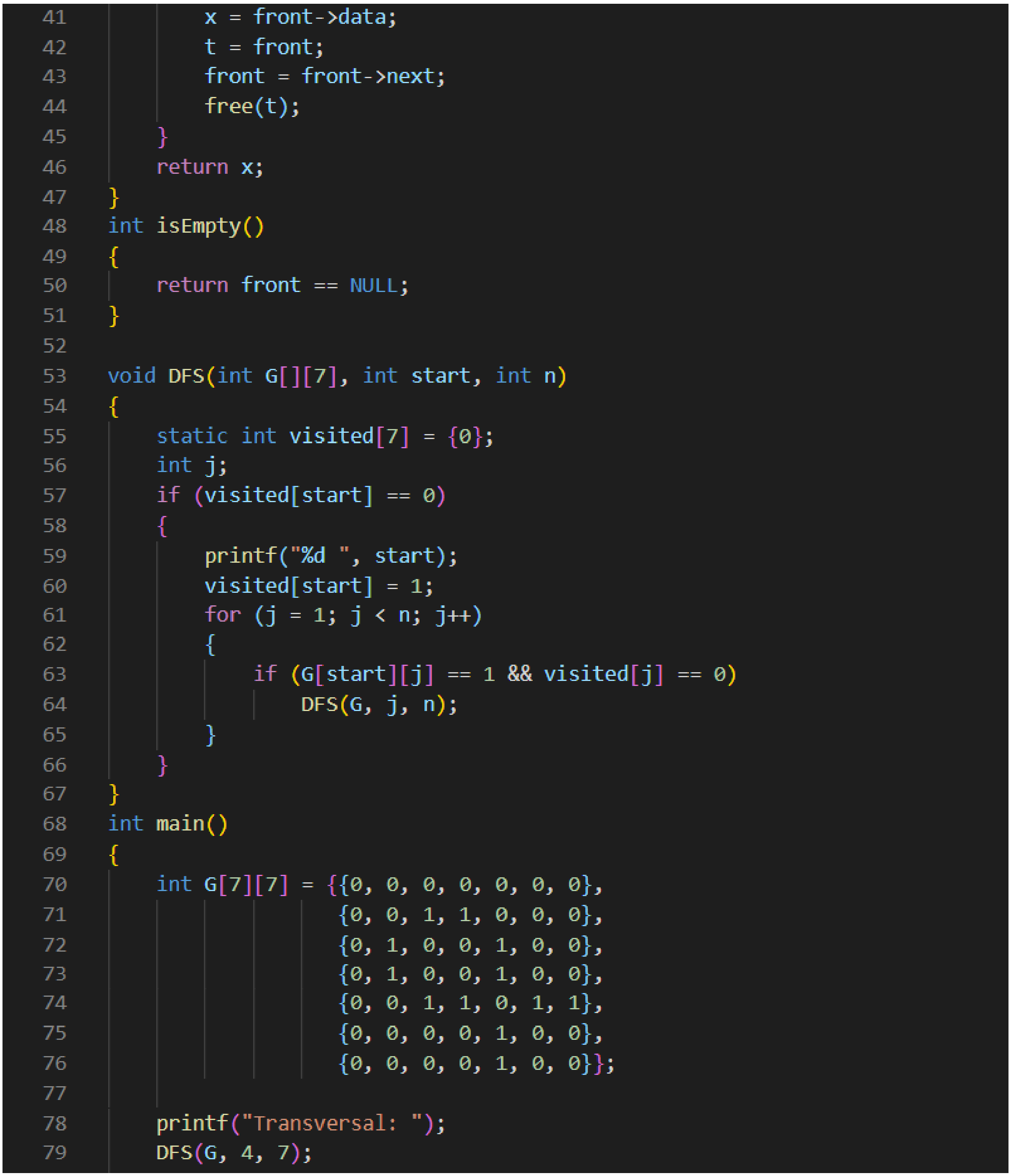
Q1. Adjacency matrix representation and DFS

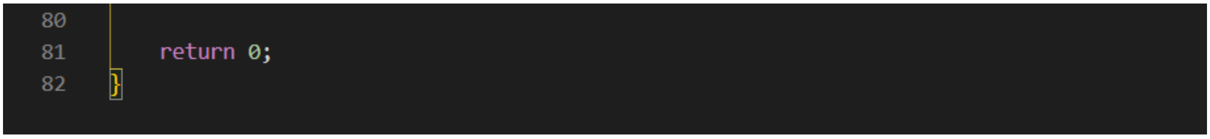
CODE:



```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

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

struct Node
{
    int data;
    struct Node *next;
} *front = NULL, *rear = NULL;
void enqueue(int x)
{
    struct Node *t;
    t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Queue is FUll\n");
    else
    {
        t->data = x;
        t->next = NULL;
        if (front == NULL)
            front = rear = t;
        else
        {
            rear->next = t;
            rear = t;
        }
    }
}
int dequeue()
{
    int x = -1;
    struct Node *t;
    if (front == NULL)
        printf("Queue is Empty\n");
    else
    {
```

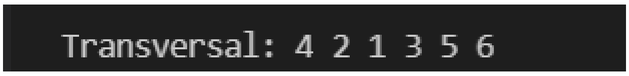


```
        x = front->data;
        t = front;
        front = front->next;
        free(t);
    }
    return x;
}
int isEmpty()
{
    return front == NULL;
}

void DFS(int G[][7], int start, int n)
{
    static int visited[7] = {0};
    int j;
    if (visited[start] == 0)
    {
        printf("%d ", start);
        visited[start] = 1;
        for (j = 1; j < n; j++)
        {
            if (G[start][j] == 1 && visited[j] == 0)
                DFS(G, j, n);
        }
    }
}
int main()
{
    int G[7][7] = {{0, 0, 0, 0, 0, 0, 0},
                   {0, 0, 1, 1, 0, 0, 0},
                   {0, 1, 0, 0, 1, 0, 0},
                   {0, 1, 0, 0, 1, 0, 0},
                   {0, 0, 1, 1, 0, 1, 1},
                   {0, 0, 0, 0, 1, 0, 0},
                   {0, 0, 0, 0, 1, 0, 0}};

    printf("Transversal: ");
    DFS(G, 4, 7);
```



```

    return 0;
}
```

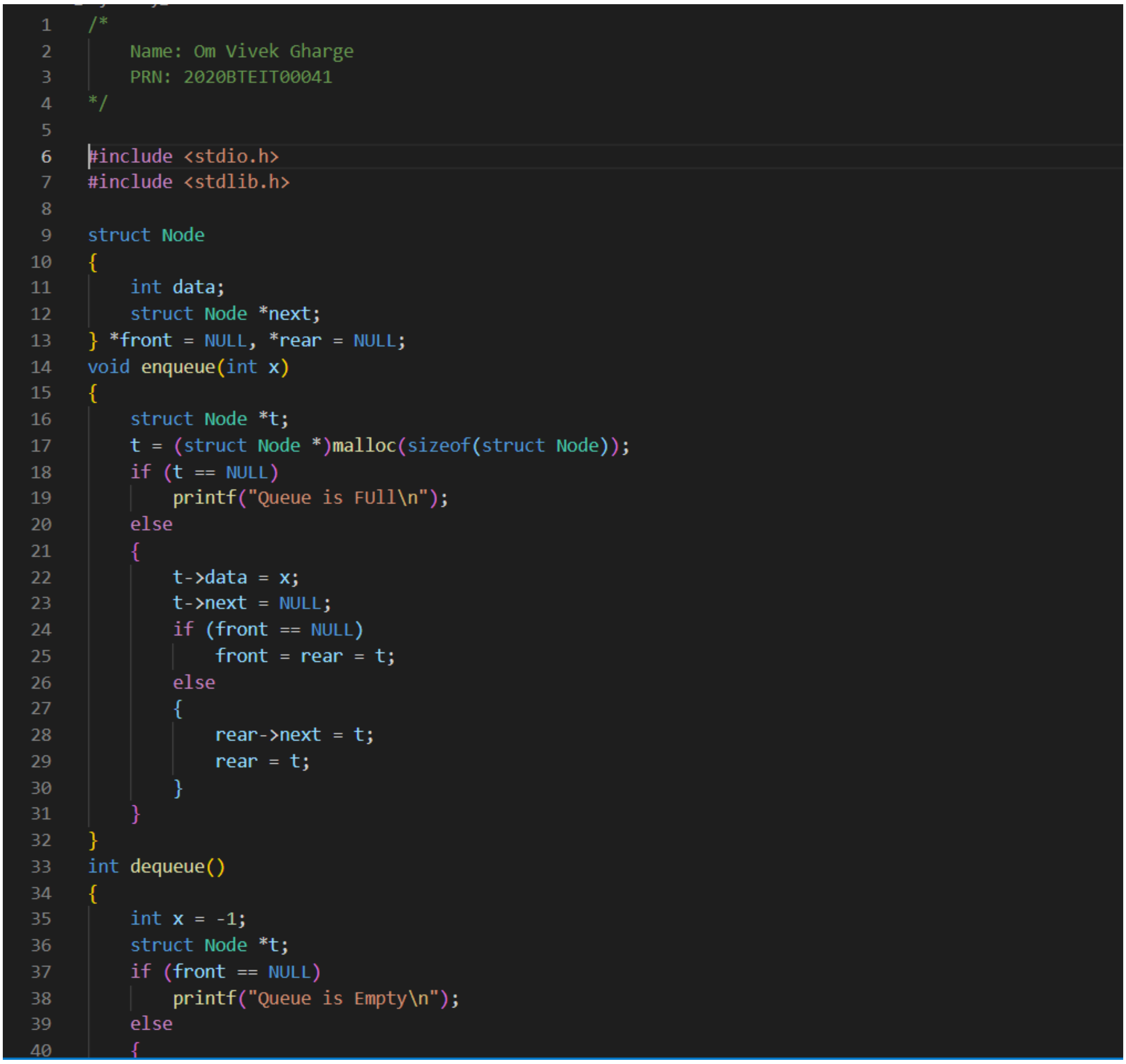
OUTPUT:



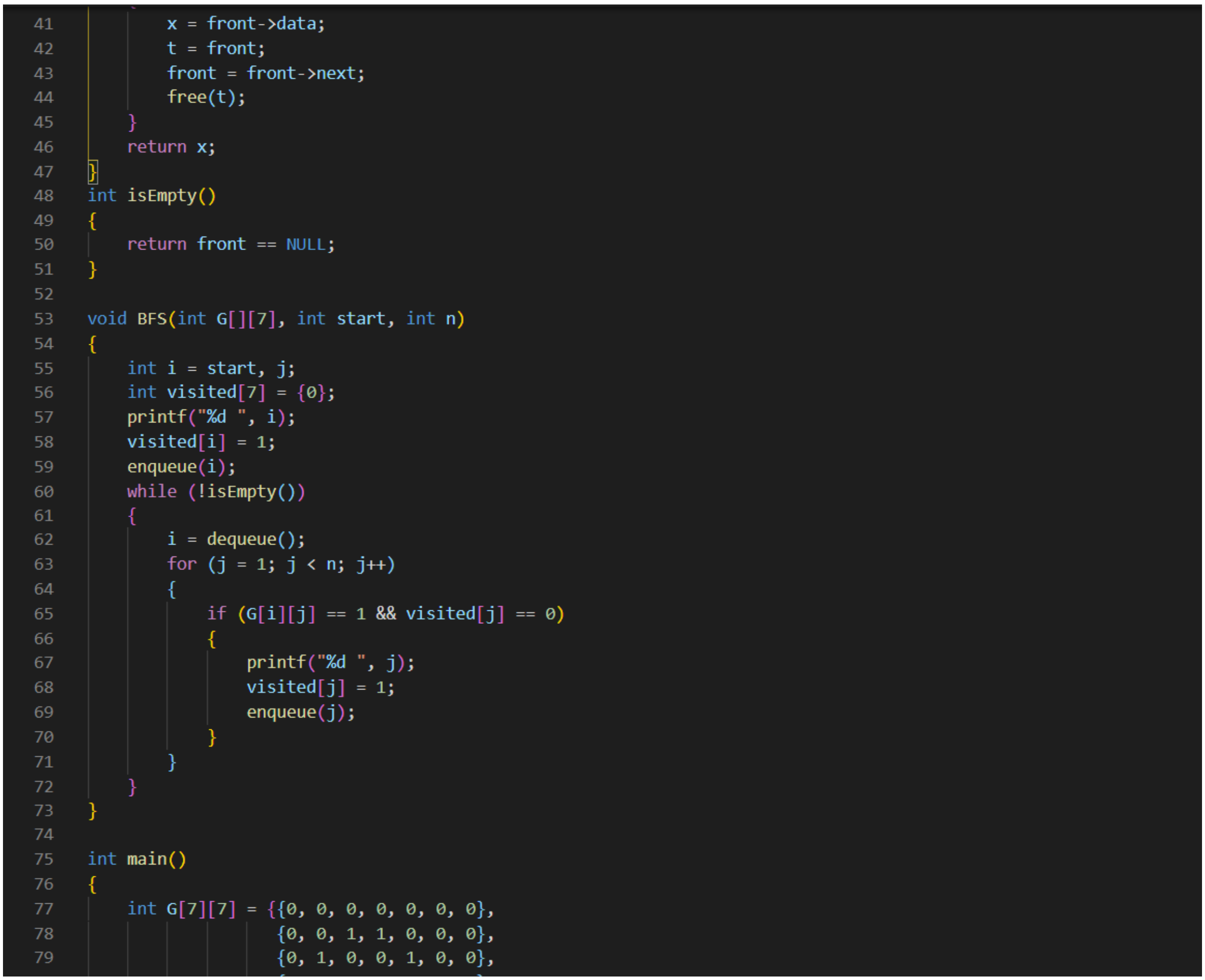
```
Transversal: 4 2 1 3 5 6
```

Q2. Adjacency matrix representation and DFS

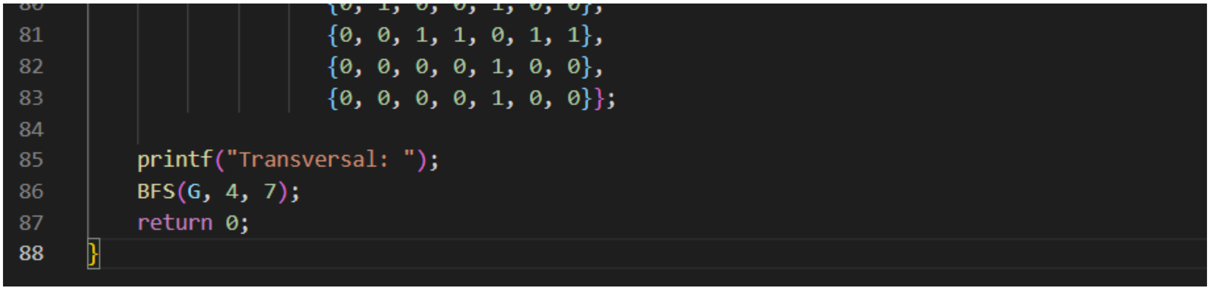
CODE:



```
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

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

struct Node
{
    int data;
    struct Node *next;
} *front = NULL, *rear = NULL;
void enqueue(int x)
{
    struct Node *t;
    t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Queue is FUll\n");
    else
    {
        t->data = x;
        t->next = NULL;
        if (front == NULL)
            front = rear = t;
        else
        {
            rear->next = t;
            rear = t;
        }
    }
}
int dequeue()
{
    int x = -1;
    struct Node *t;
    if (front == NULL)
        printf("Queue is Empty\n");
    else
    {
```

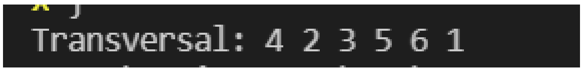


```c
        x = front->data;
        t = front;
        front = front->next;
        free(t);
    }
    return x;
}
int isEmpty()
{
    return front == NULL;
}

void BFS(int G[][7], int start, int n)
{
    int i = start, j;
    int visited[7] = {0};
    printf("%d ", i);
    visited[i] = 1;
    enqueue(i);
    while (!isEmpty())
    {
        i = dequeue();
        for (j = 1; j < n; j++)
        {
            if (G[i][j] == 1 && visited[j] == 0)
            {
                printf("%d ", j);
                visited[j] = 1;
                enqueue(j);
            }
        }
    }
}

int main()
{
    int G[7][7] = {{0, 0, 0, 0, 0, 0, 0},
                   {0, 0, 1, 1, 0, 0, 0},
                   {0, 1, 0, 0, 1, 0, 0},
```



```c
                   {0, 1, 0, 0, 1, 0, 0},
                   {0, 0, 1, 1, 0, 1, 1},
                   {0, 0, 0, 0, 1, 0, 0},
                   {0, 0, 0, 0, 1, 0, 0}};

    printf("Transversal: ");
    BFS(G, 4, 7);
    return 0;
}
```

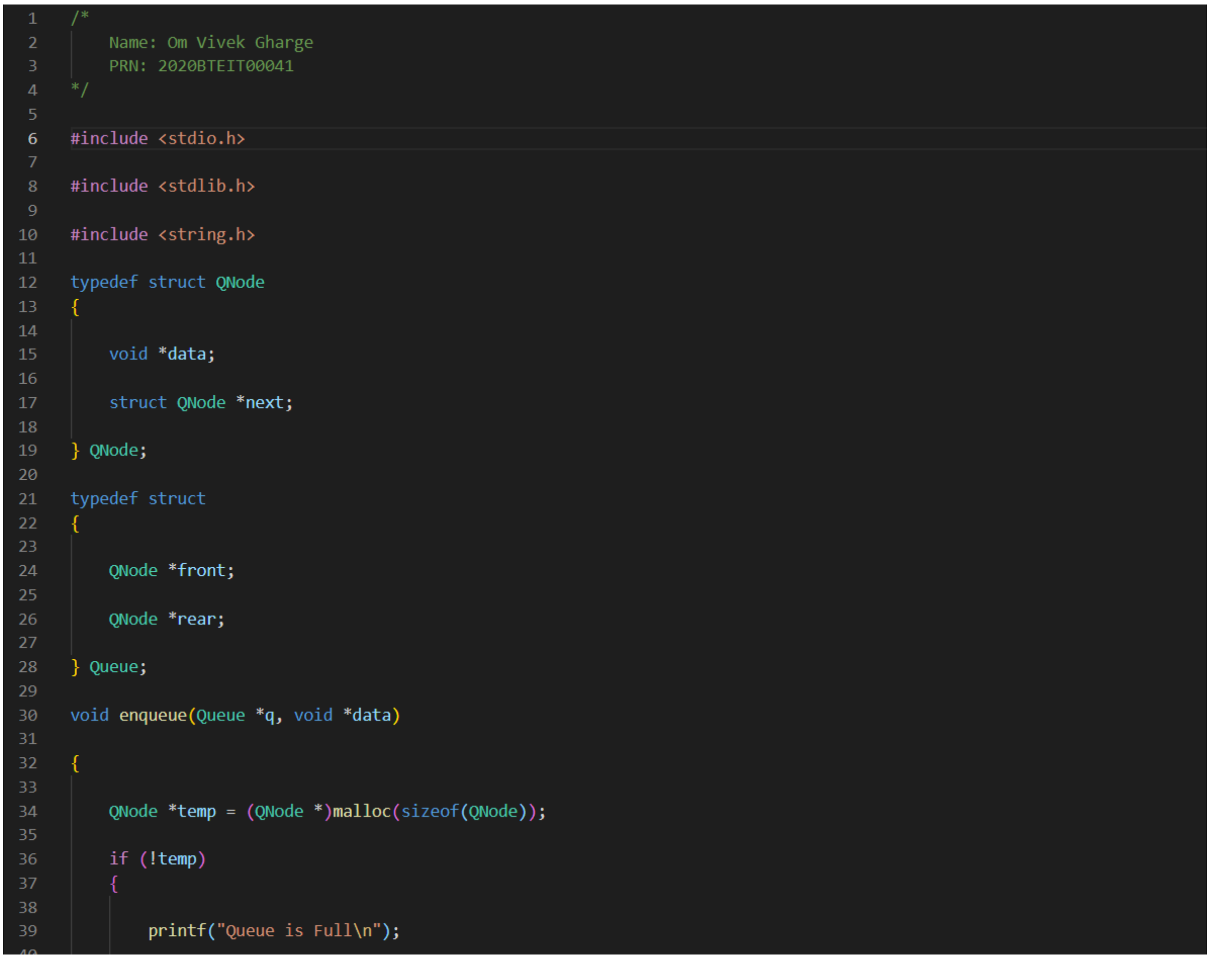
CODE:



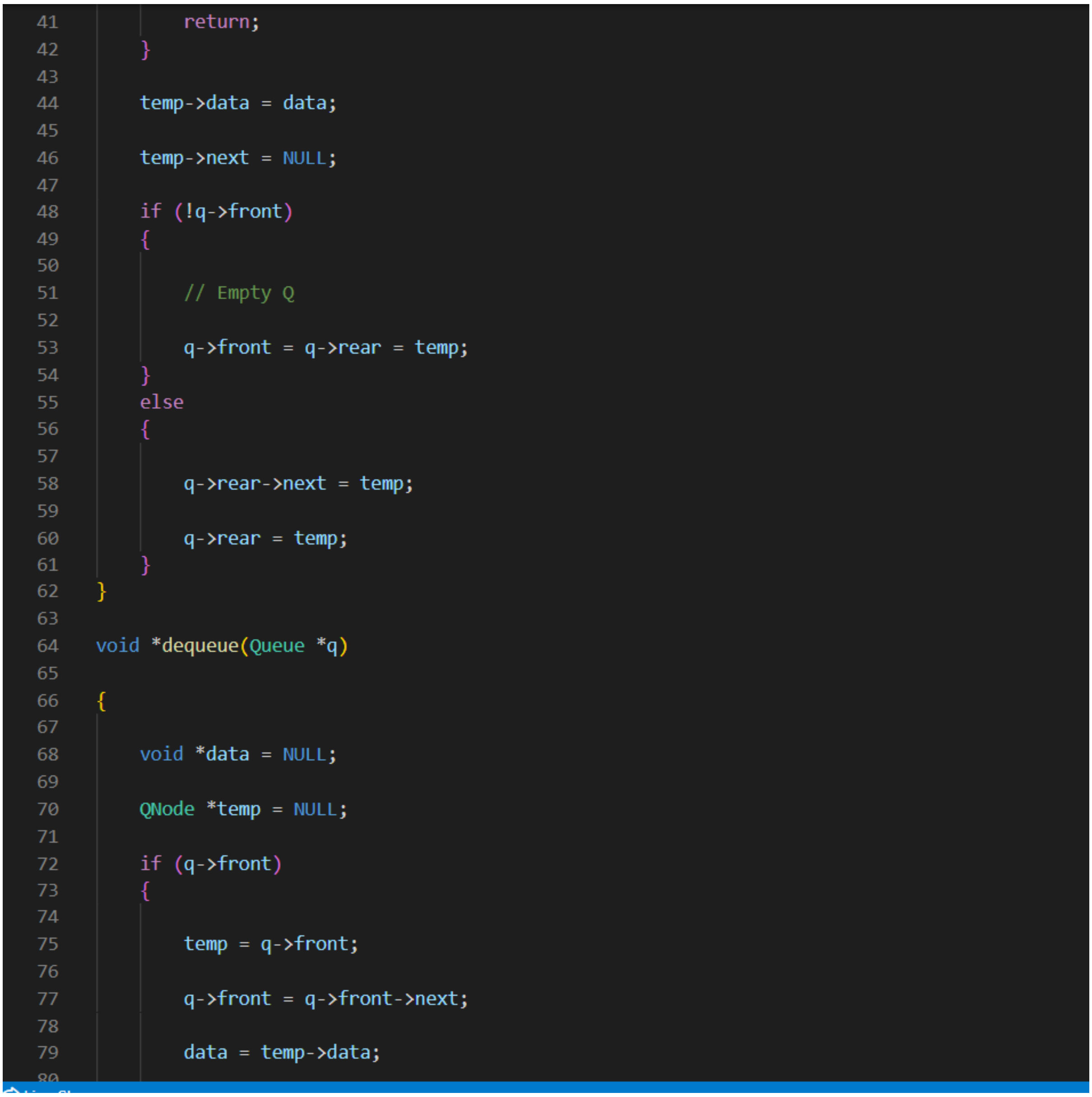
```
Transversal: 4 2 3 5 6 1
```

Q3. Adjacency list representation and DFS

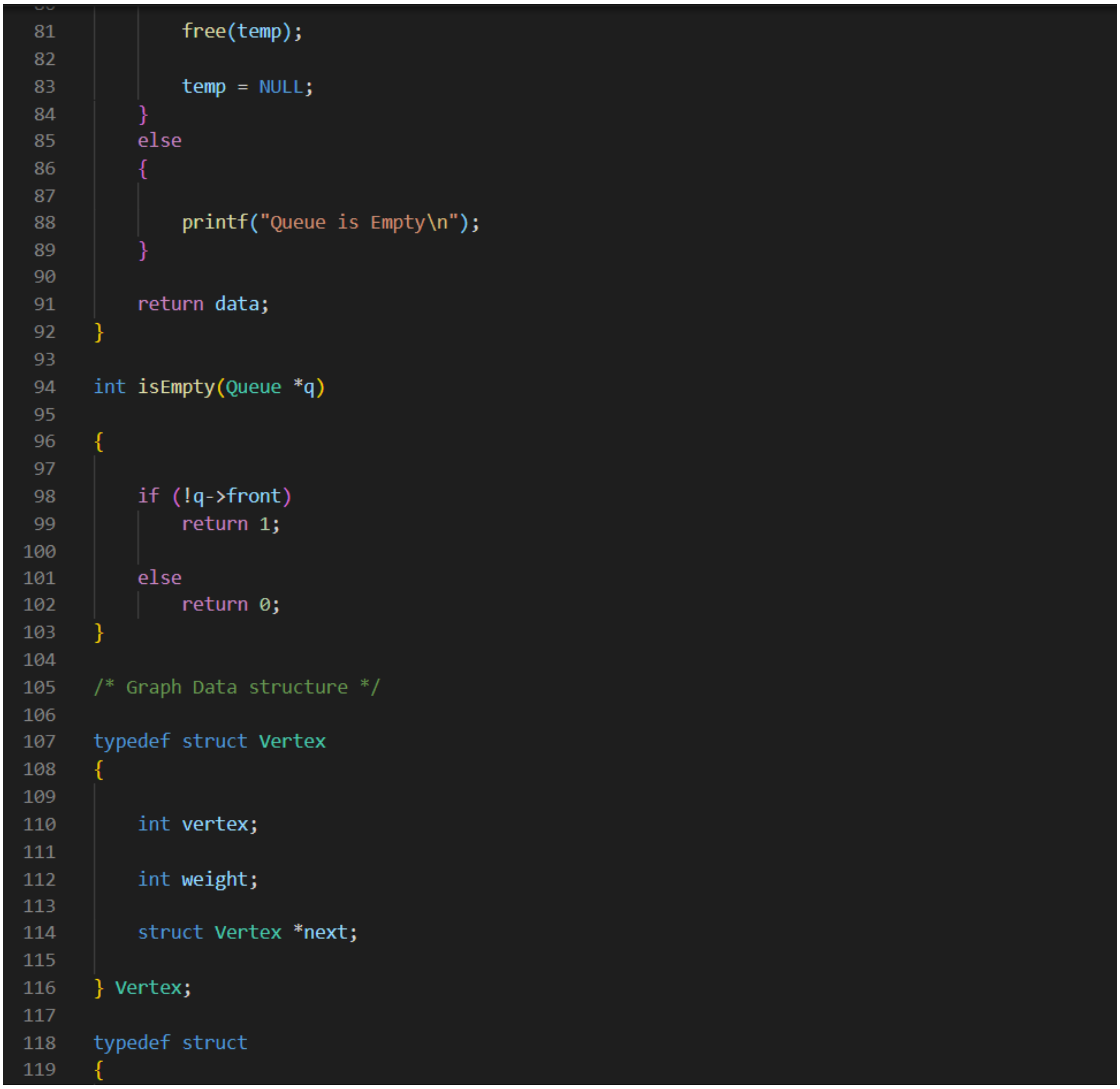
CODE:



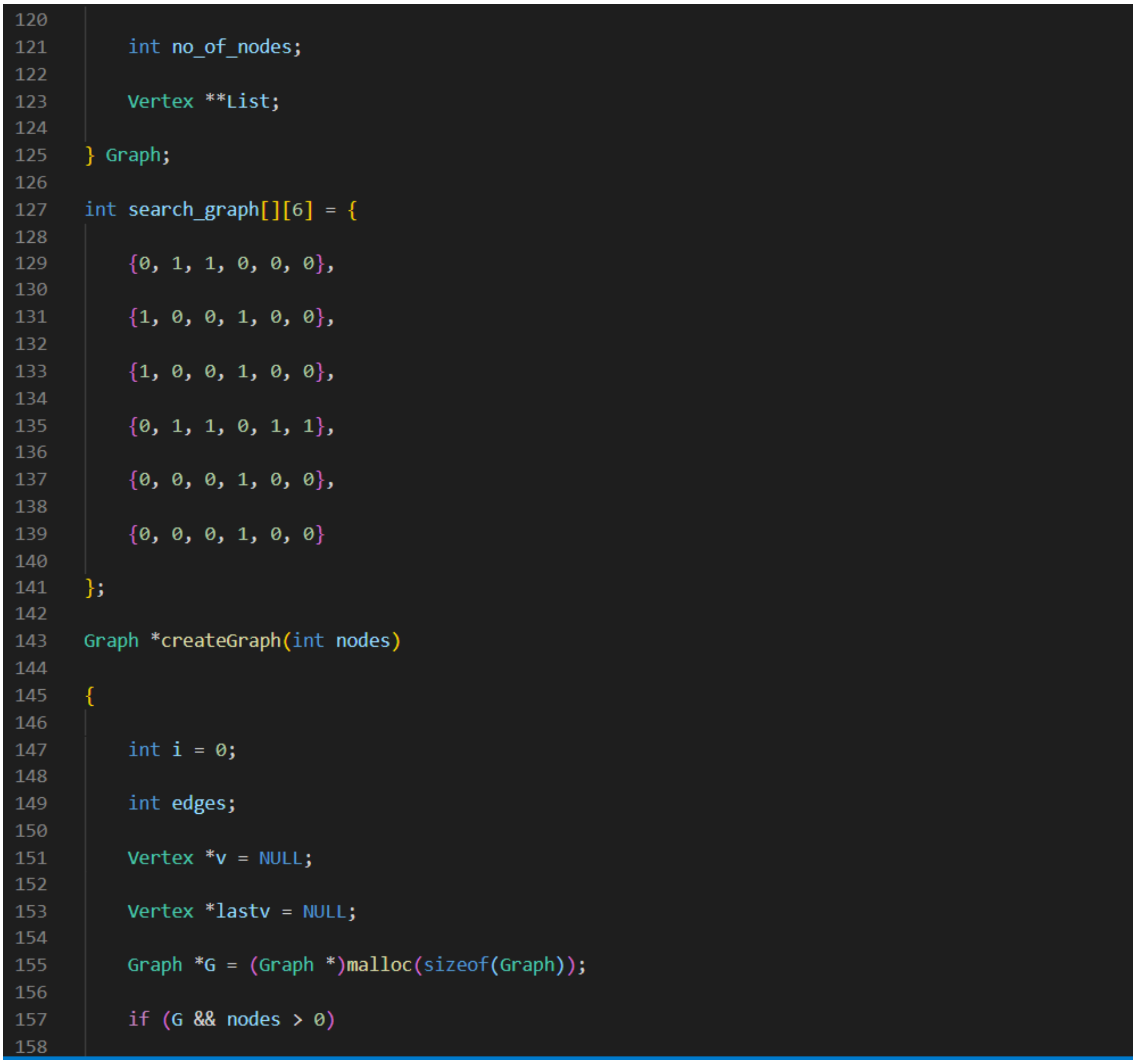
```
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct QNode
{

    void *data;

    struct QNode *next;

} QNode;

typedef struct
{

    QNode *front;

    QNode *rear;

} Queue;

void enqueue(Queue *q, void *data)

{

    QNode *temp = (QNode *)malloc(sizeof(QNode));

    if (!temp)
    {

        printf("Queue is Full\n");
```



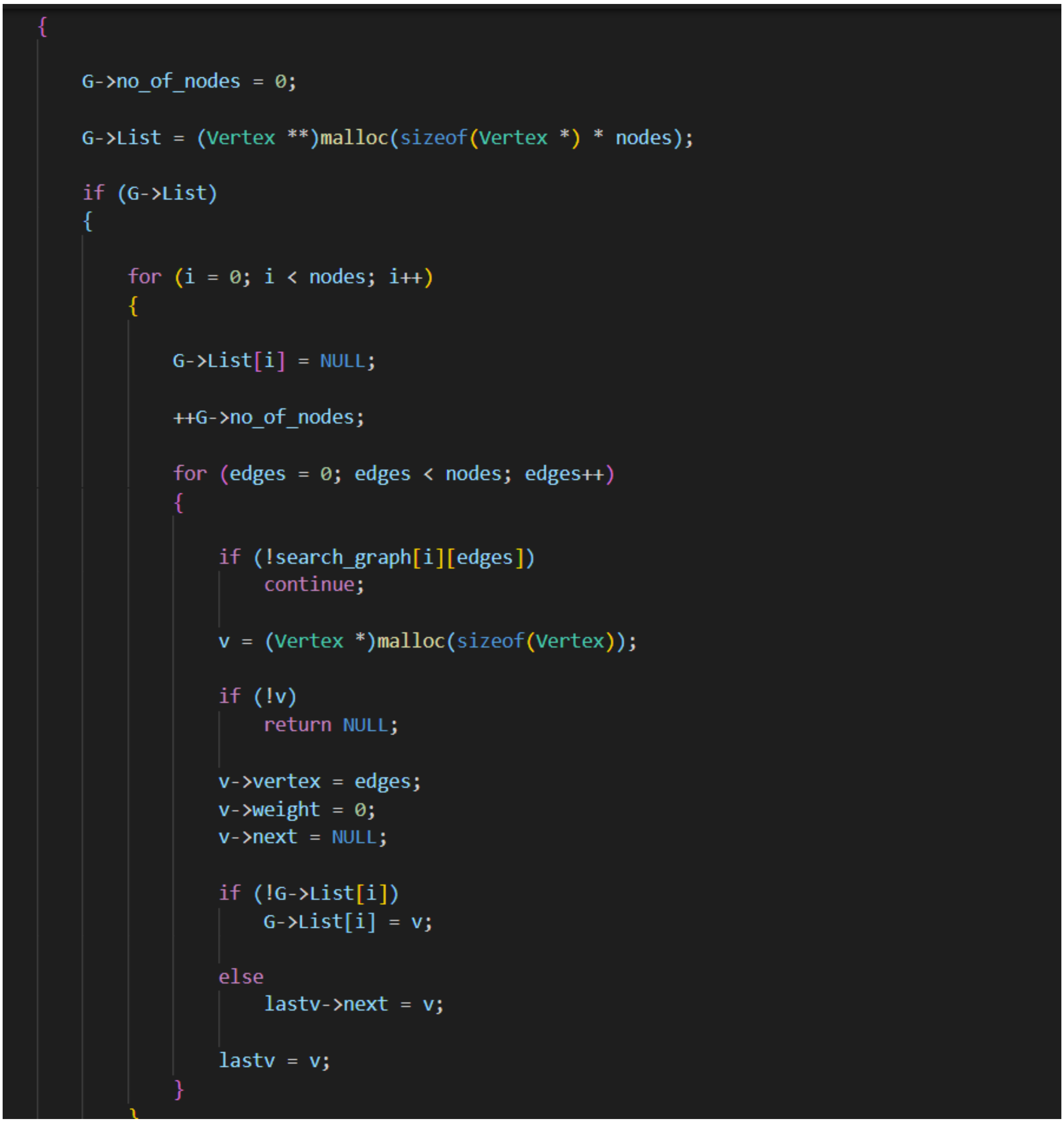
```
        return;
    }

    temp->data = data;

    temp->next = NULL;

    if (!q->front)
    {

        // Empty Q

        q->front = q->rear = temp;
    }
    else
    {

        q->rear->next = temp;

        q->rear = temp;
    }
}

void *dequeue(Queue *q)

{

    void *data = NULL;

    QNode *temp = NULL;

    if (q->front)
    {

        temp = q->front;

        q->front = q->front->next;

        data = temp->data;
```



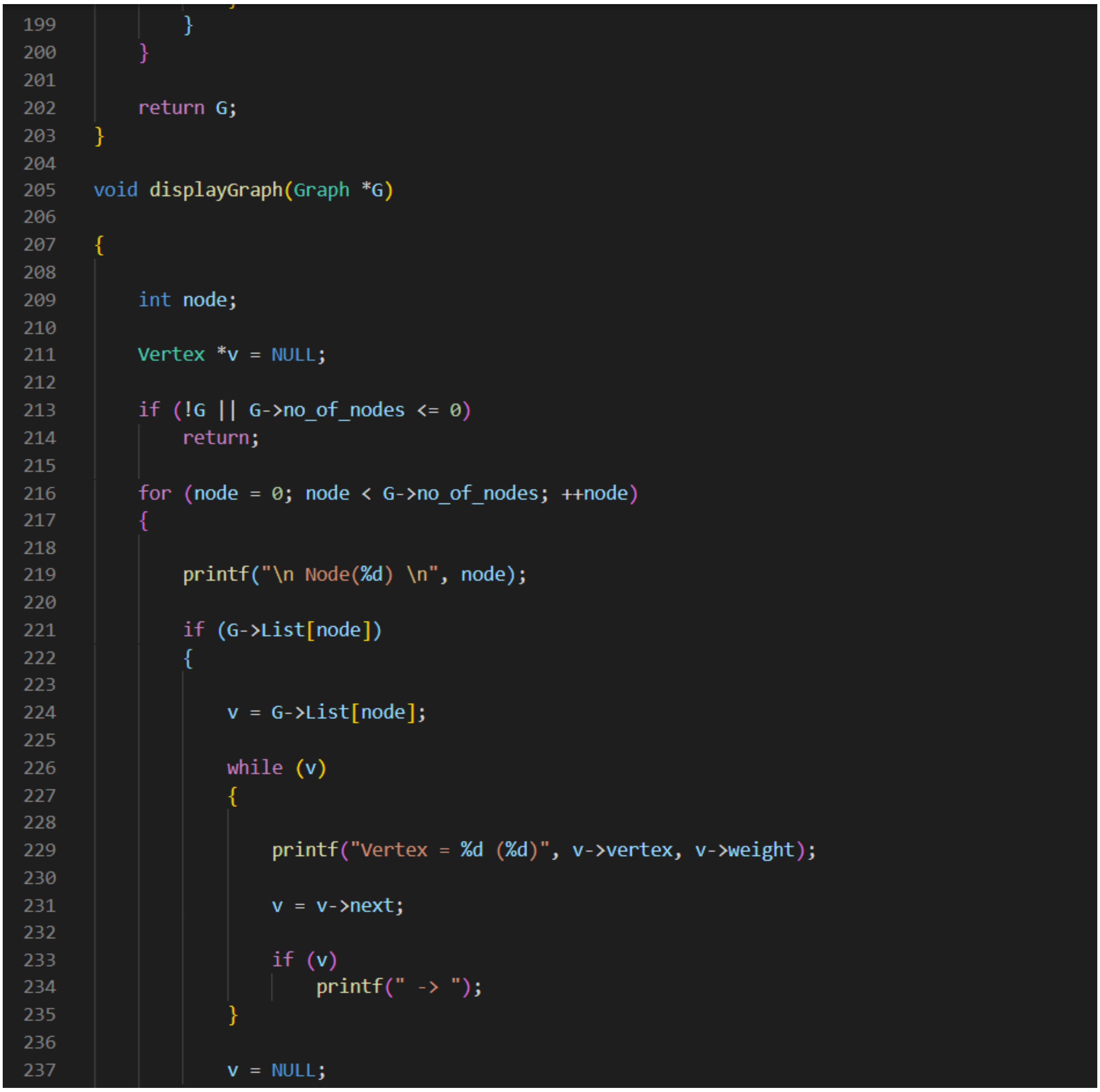
```
        free(temp);

        temp = NULL;
    }
    else
    {

        printf("Queue is Empty\n");
    }

    return data;
}

int isEmpty(Queue *q)

{

    if (!q->front)
        return 1;

    else
        return 0;
}

/* Graph Data structure */

typedef struct Vertex
{

    int vertex;

    int weight;

    struct Vertex *next;

} Vertex;

typedef struct
{
```



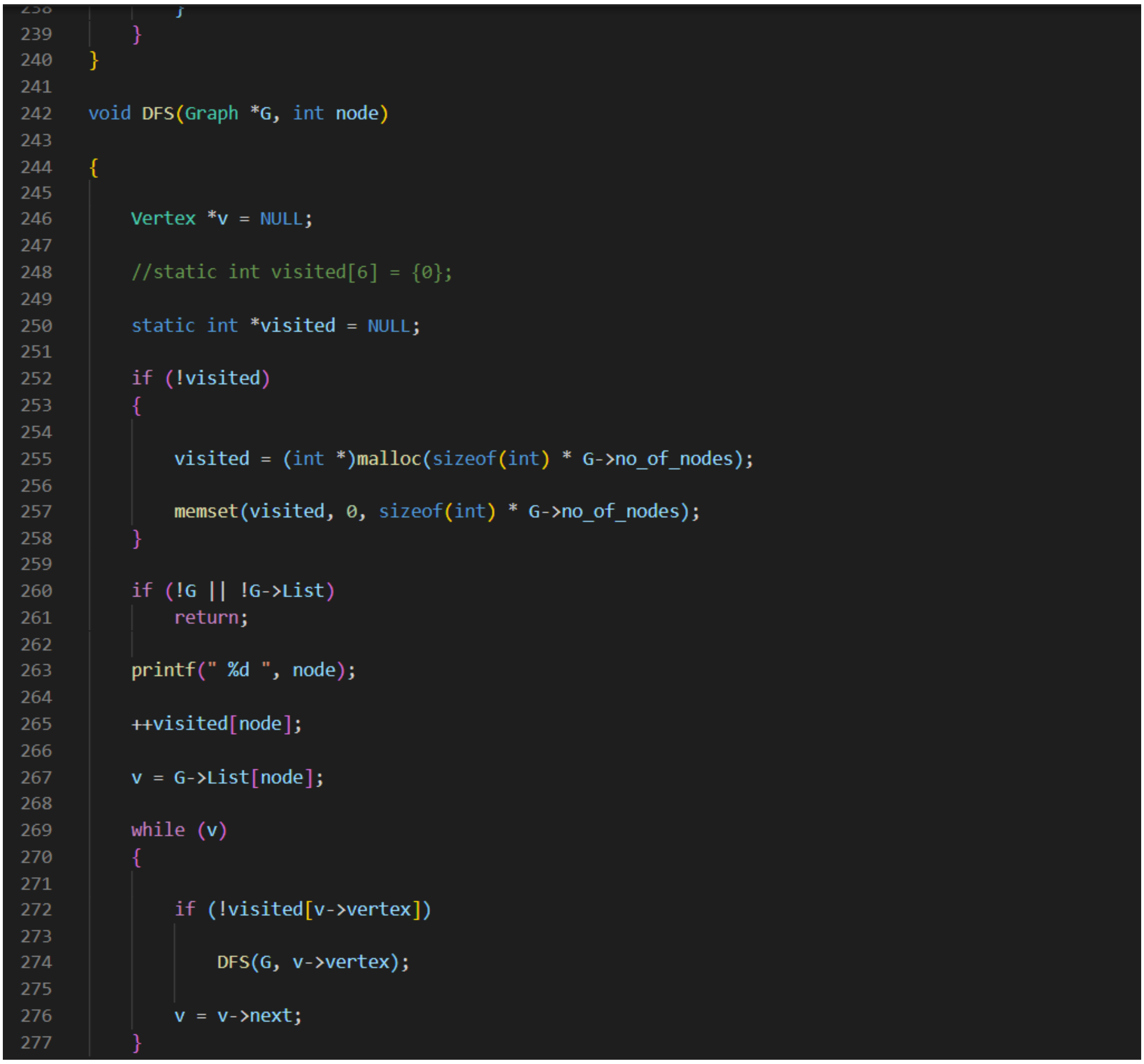
```

    int no_of_nodes;

    Vertex **List;

} Graph;

int search_graph[][6] = {

    {0, 1, 1, 0, 0, 0},

    {1, 0, 0, 1, 0, 0},

    {1, 0, 0, 1, 0, 0},

    {0, 1, 1, 0, 1, 1},

    {0, 0, 0, 1, 0, 0},

    {0, 0, 0, 1, 0, 0}

};

Graph *createGraph(int nodes)

{

    int i = 0;

    int edges;

    Vertex *v = NULL;

    Vertex *lastv = NULL;

    Graph *G = (Graph *)malloc(sizeof(Graph));

    if (G && nodes > 0)

```



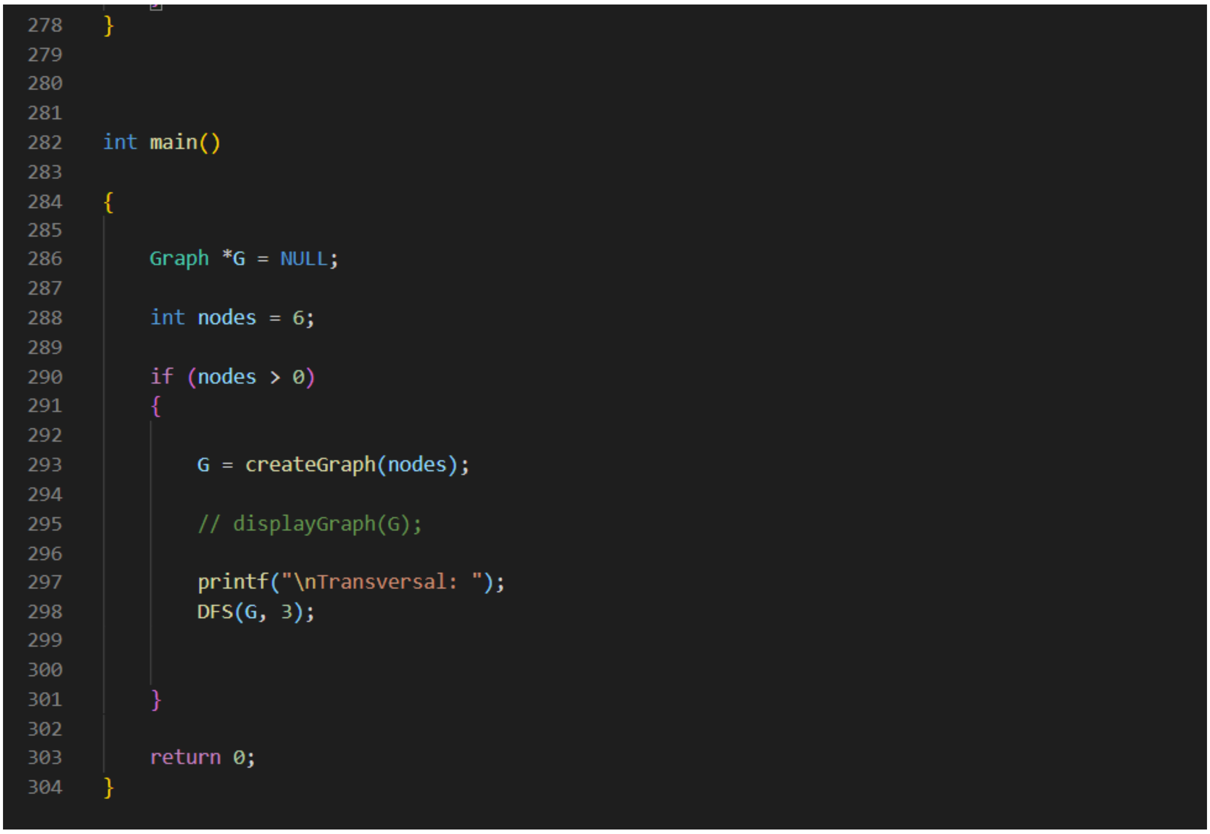
```
{

    G->no_of_nodes = 0;

    G->List = (Vertex **)malloc(sizeof(Vertex *) * nodes);

    if (G->List)
    {

        for (i = 0; i < nodes; i++)
        {

            G->List[i] = NULL;

            ++G->no_of_nodes;

            for (edges = 0; edges < nodes; edges++)
            {

                if (!search_graph[i][edges])
                    continue;

                v = (Vertex *)malloc(sizeof(Vertex));

                if (!v)
                    return NULL;

                v->vertex = edges;
                v->weight = 0;
                v->next = NULL;

                if (!G->List[i])
                    G->List[i] = v;

                else
                    lastv->next = v;

                lastv = v;
            }
        }
```



```
        }
    }

    return G;
}

void displayGraph(Graph *G)

{

    int node;

    Vertex *v = NULL;

    if (!G || G->no_of_nodes <= 0)
        return;

    for (node = 0; node < G->no_of_nodes; ++node)
    {

        printf("\n Node(%d) \n", node);

        if (G->List[node])
        {

            v = G->List[node];

            while (v)
            {

                printf("Vertex = %d (%d)", v->vertex, v->weight);

                v = v->next;

                if (v)
                    printf(" -> ");
            }

            v = NULL;
```

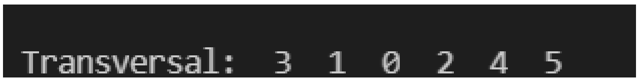


```c
        }
    }
}

void DFS(Graph *G, int node)

{

    Vertex *v = NULL;

    //static int visited[6] = {0};

    static int *visited = NULL;

    if (!visited)
    {

        visited = (int *)malloc(sizeof(int) * G->no_of_nodes);

        memset(visited, 0, sizeof(int) * G->no_of_nodes);
    }

    if (!G || !G->List)
        return;

    printf(" %d ", node);

    ++visited[node];

    v = G->List[node];

    while (v)
    {

        if (!visited[v->vertex])

            DFS(G, v->vertex);

        v = v->next;
    }
```



```
}



int main()

{

    Graph *G = NULL;

    int nodes = 6;

    if (nodes > 0)
    {

        G = createGraph(nodes);

        // displayGraph(G);

        printf("\nTransversal: ");
        DFS(G, 3);


    }

    return 0;
}
```

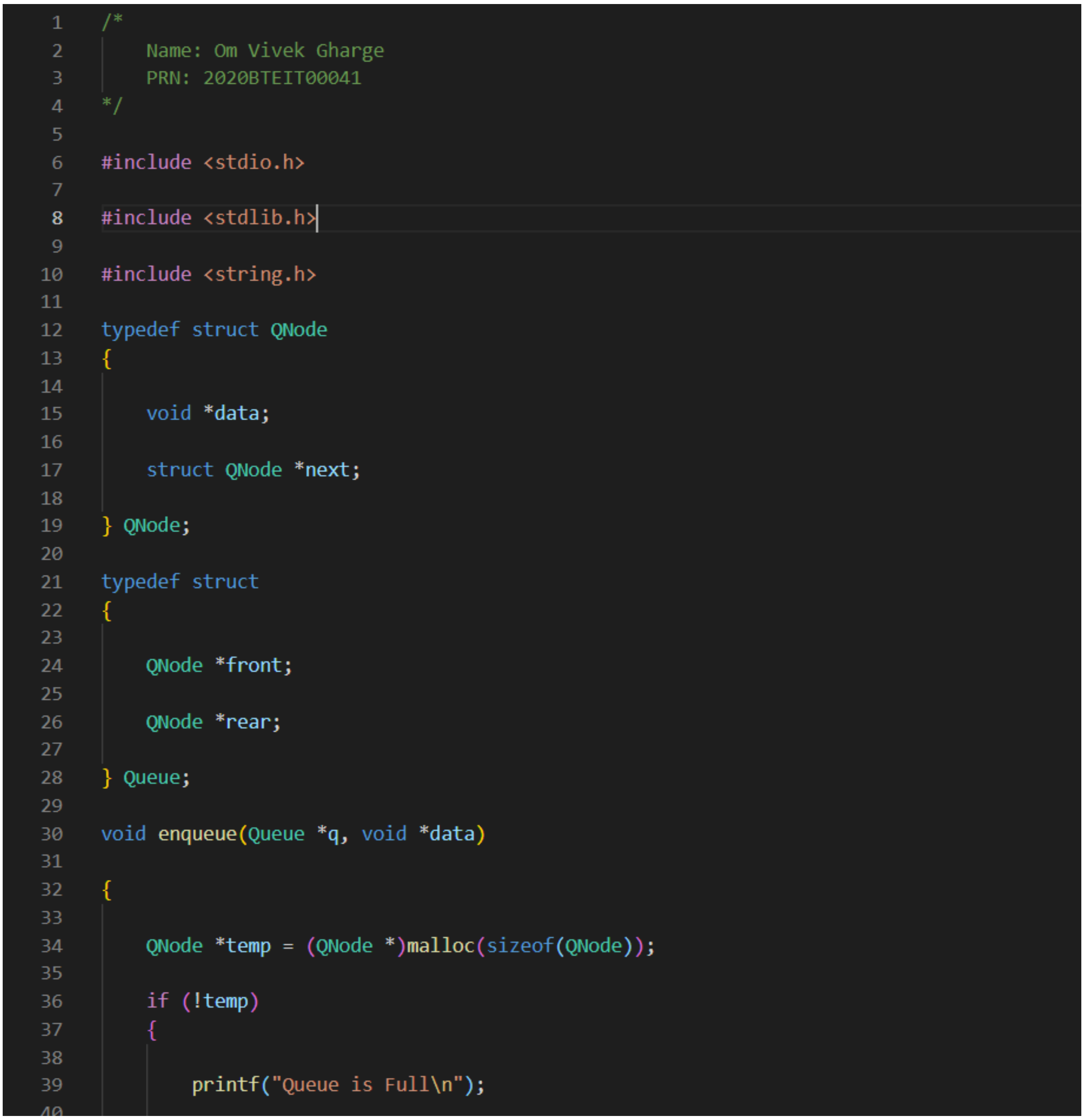
OUTPUT:



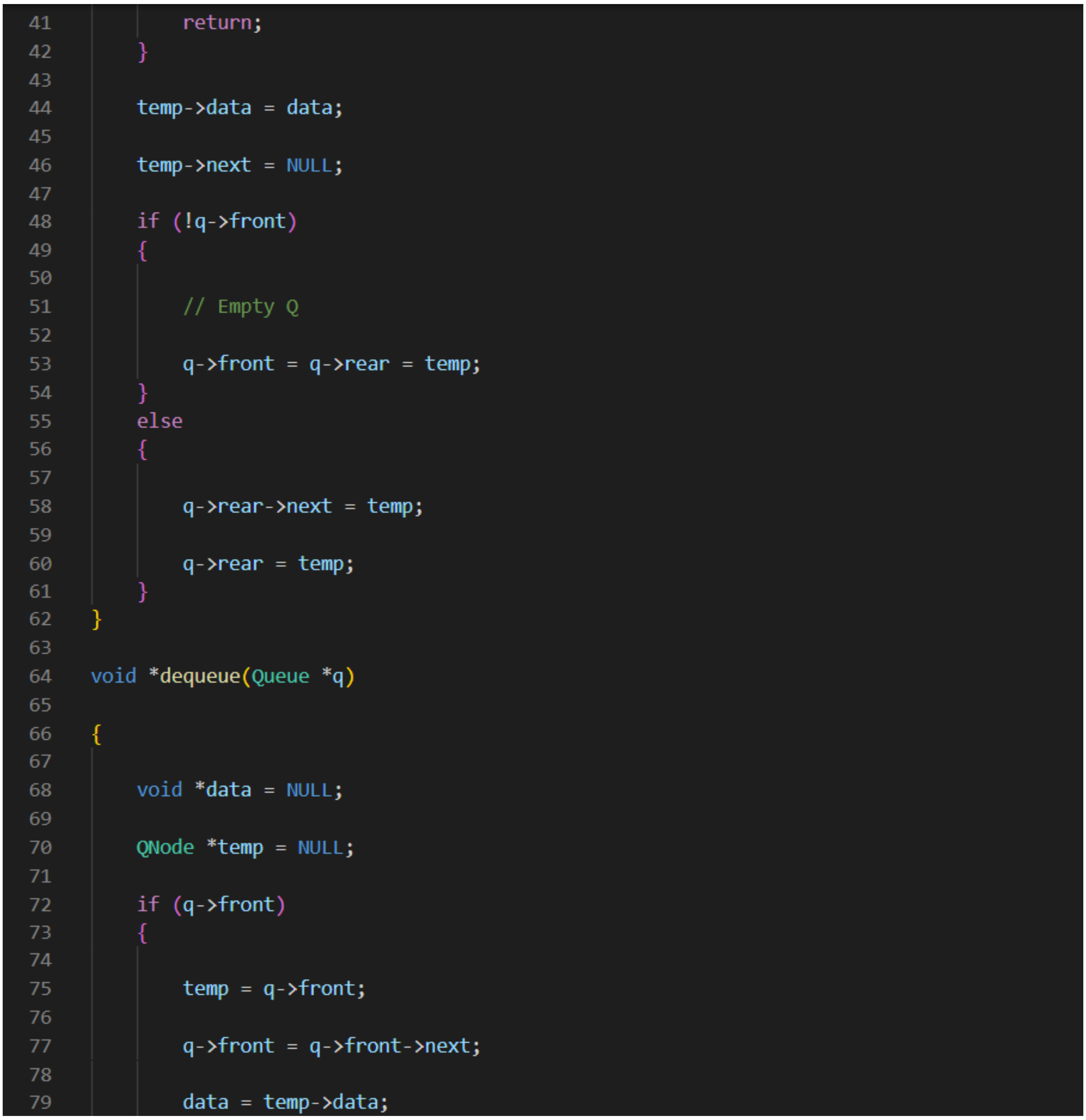
```
Transversal:  3  1  0  2  4  5
```

Q4. Adjacency list representation and BFS

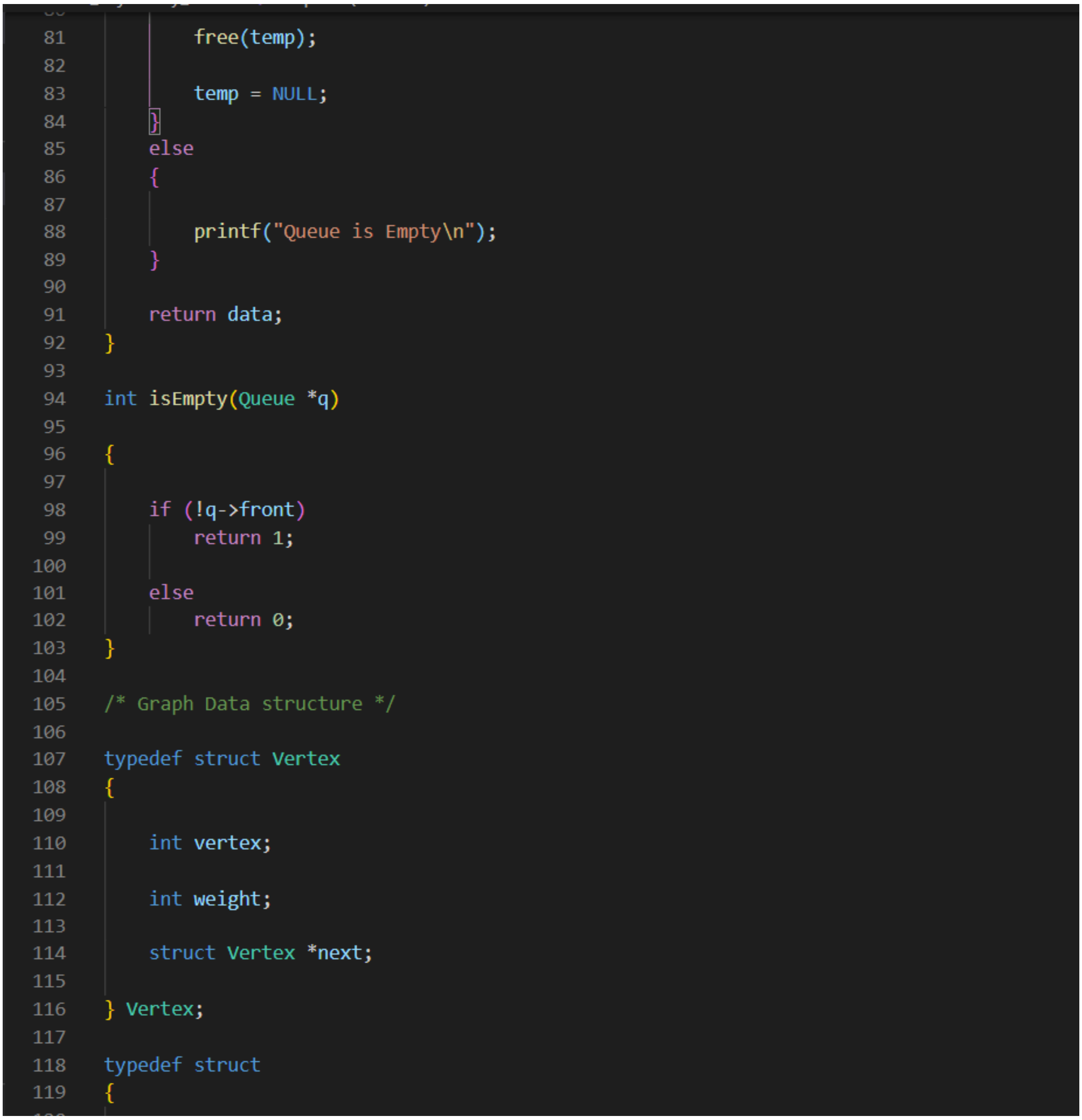
CODE:



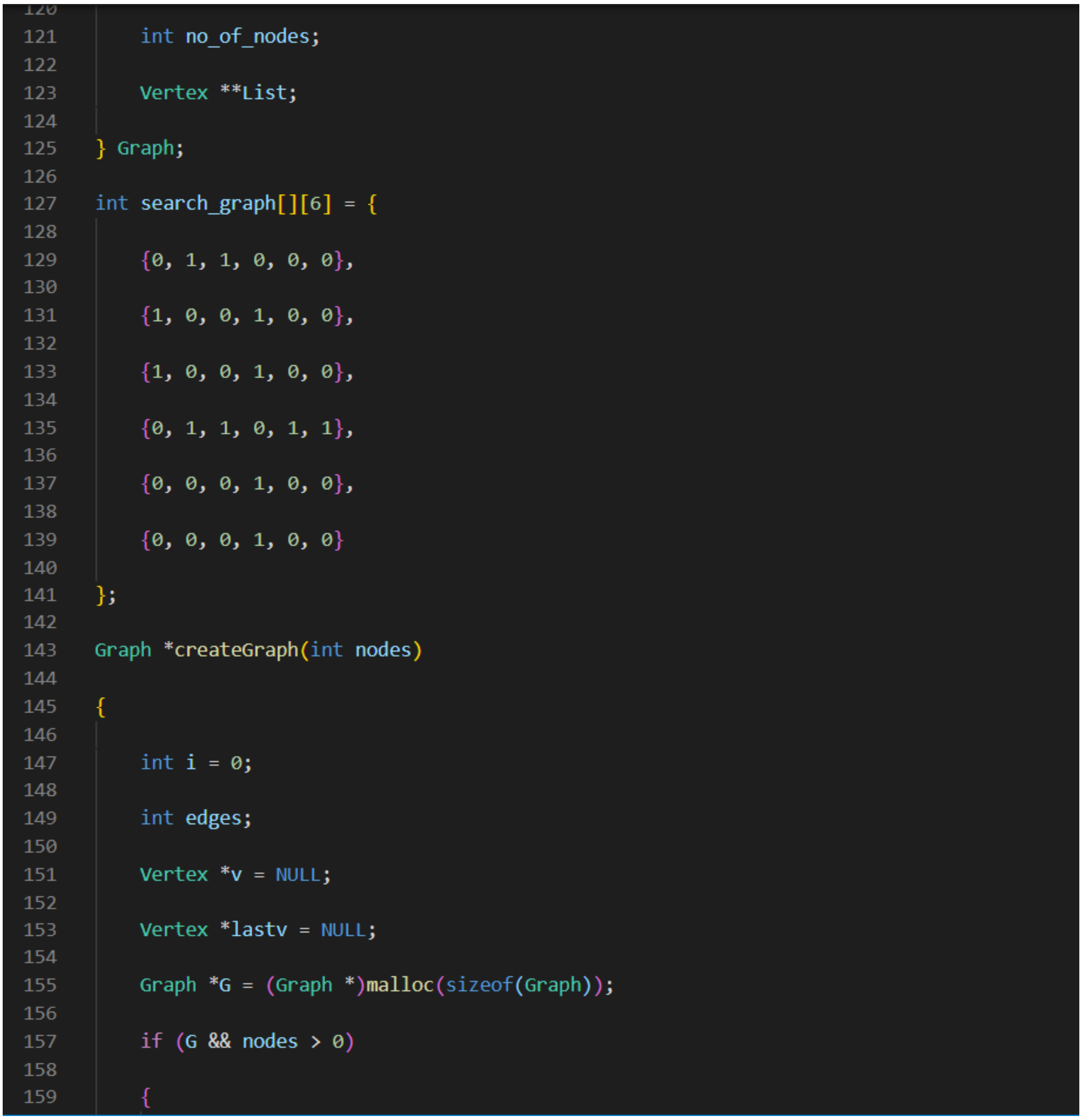
```
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct QNode
{

    void *data;

    struct QNode *next;

} QNode;

typedef struct
{

    QNode *front;

    QNode *rear;

} Queue;

void enqueue(Queue *q, void *data)

{

    QNode *temp = (QNode *)malloc(sizeof(QNode));

    if (!temp)
    {

        printf("Queue is Full\n");

```



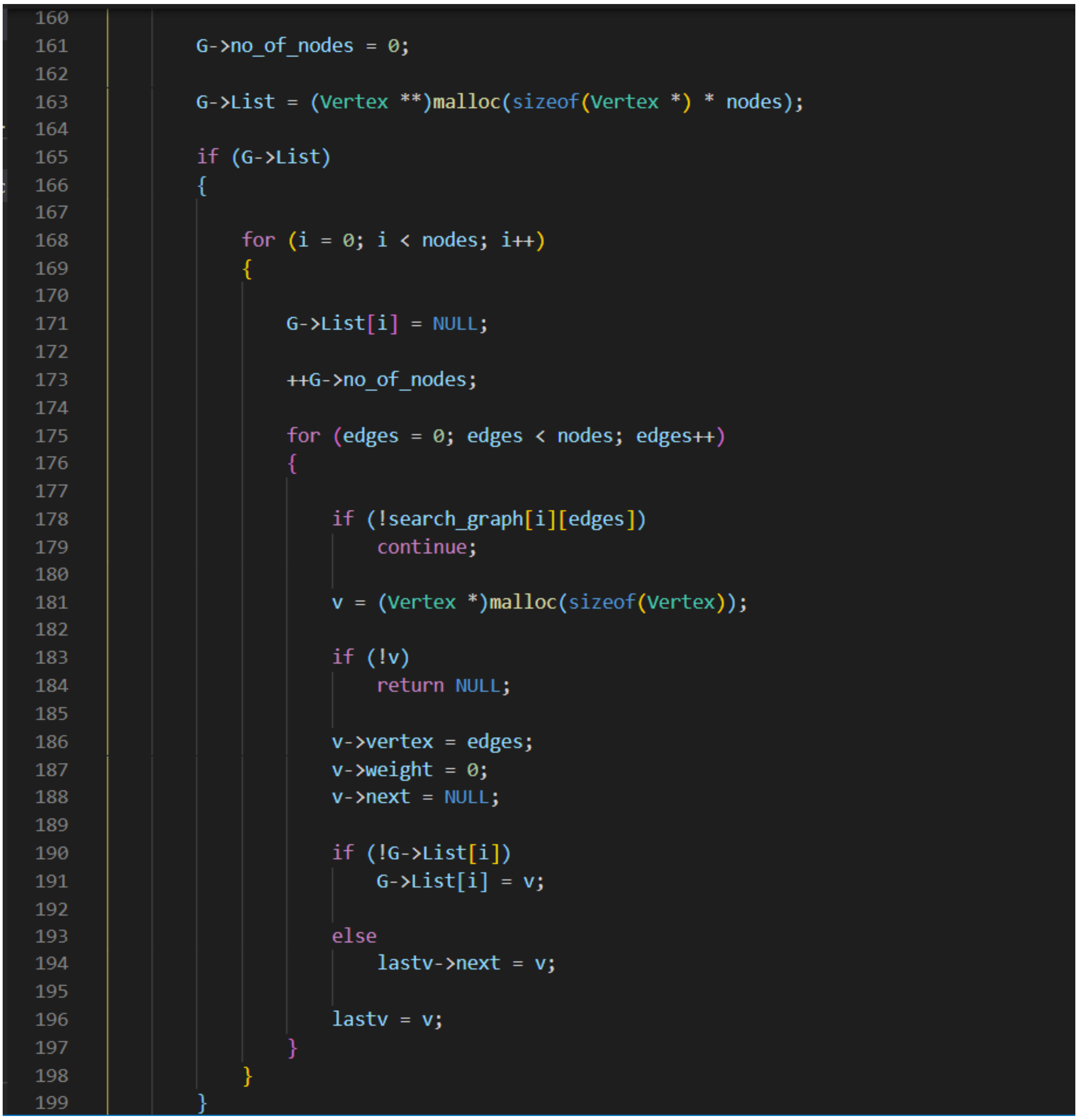
```
        return;
    }

    temp->data = data;

    temp->next = NULL;

    if (!q->front)
    {

        // Empty Q

        q->front = q->rear = temp;
    }
    else
    {

        q->rear->next = temp;

        q->rear = temp;
    }
}

void *dequeue(Queue *q)

{

    void *data = NULL;

    QNode *temp = NULL;

    if (q->front)
    {

        temp = q->front;

        q->front = q->front->next;

        data = temp->data;
```



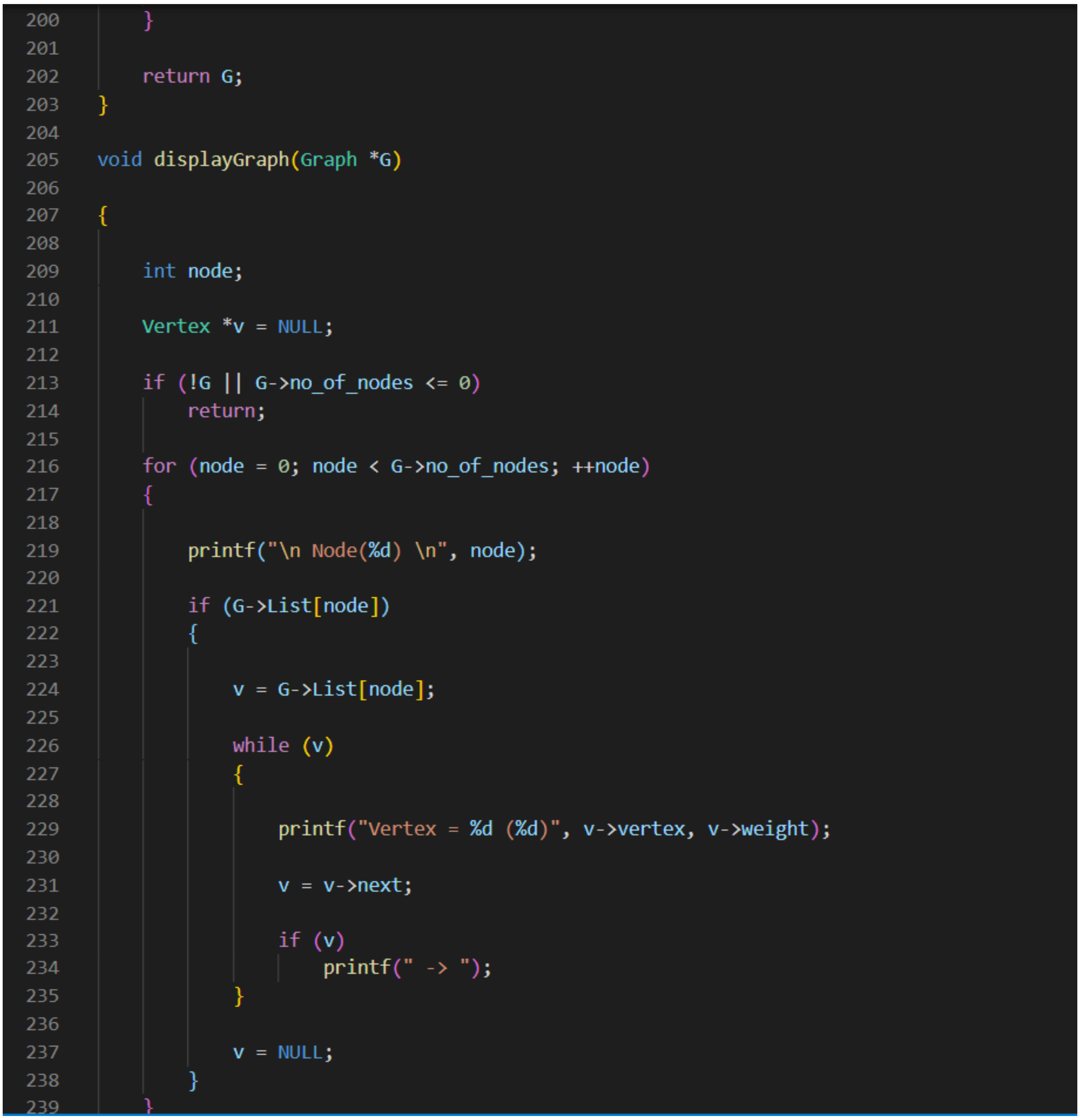
```
        free(temp);

        temp = NULL;
    }
    else
    {

        printf("Queue is Empty\n");
    }

    return data;
}

int isEmpty(Queue *q)

{

    if (!q->front)
        return 1;

    else
        return 0;
}

/* Graph Data structure */

typedef struct Vertex
{

    int vertex;

    int weight;

    struct Vertex *next;

} Vertex;

typedef struct
{
```



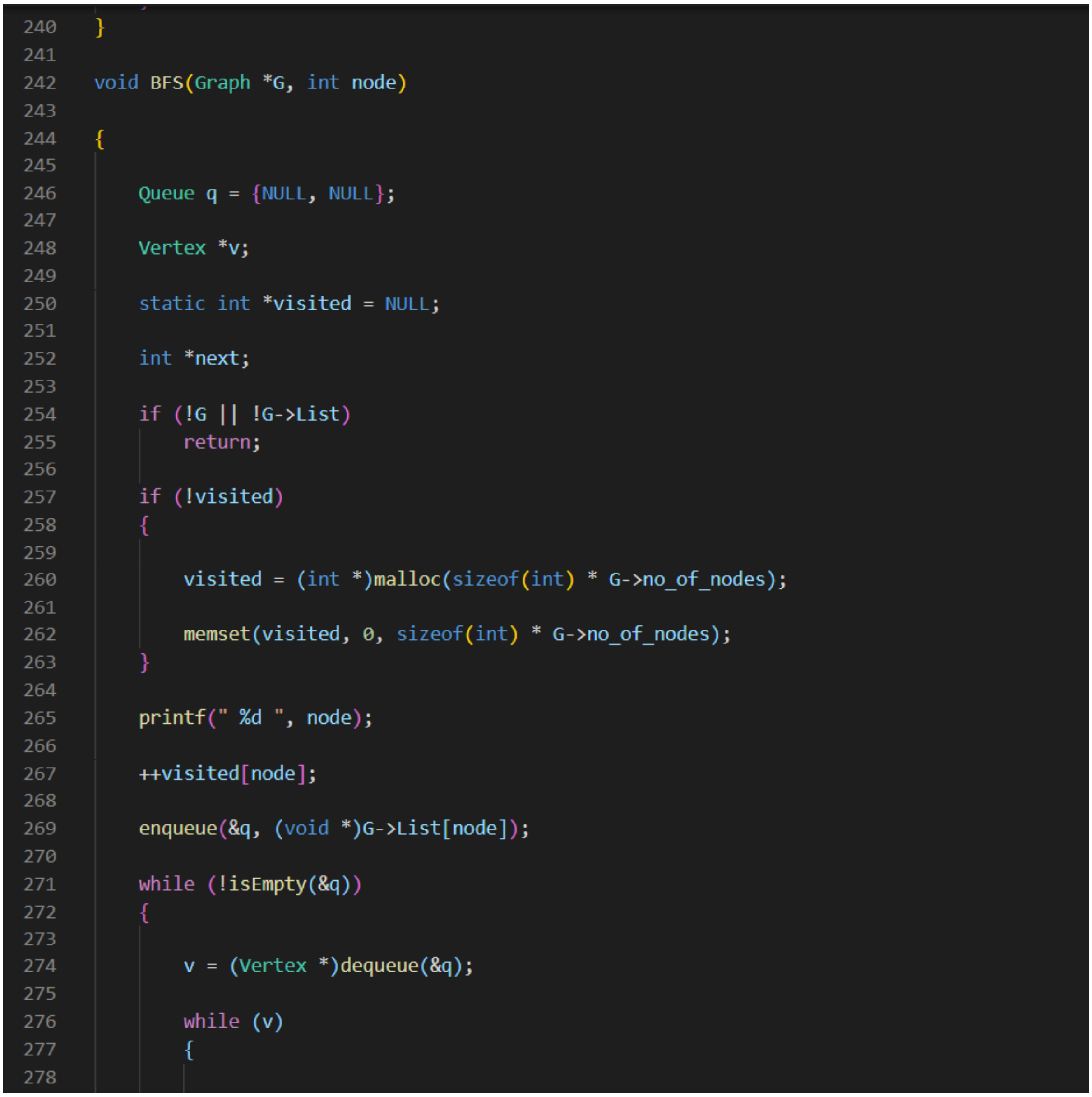
```
    int no_of_nodes;

    Vertex **List;

} Graph;

int search_graph[][6] = {

    {0, 1, 1, 0, 0, 0},

    {1, 0, 0, 1, 0, 0},

    {1, 0, 0, 1, 0, 0},

    {0, 1, 1, 0, 1, 1},

    {0, 0, 0, 1, 0, 0},

    {0, 0, 0, 1, 0, 0}

};

Graph *createGraph(int nodes)

{

    int i = 0;

    int edges;

    Vertex *v = NULL;

    Vertex *lastv = NULL;

    Graph *G = (Graph *)malloc(sizeof(Graph));

    if (G && nodes > 0)

    {
```



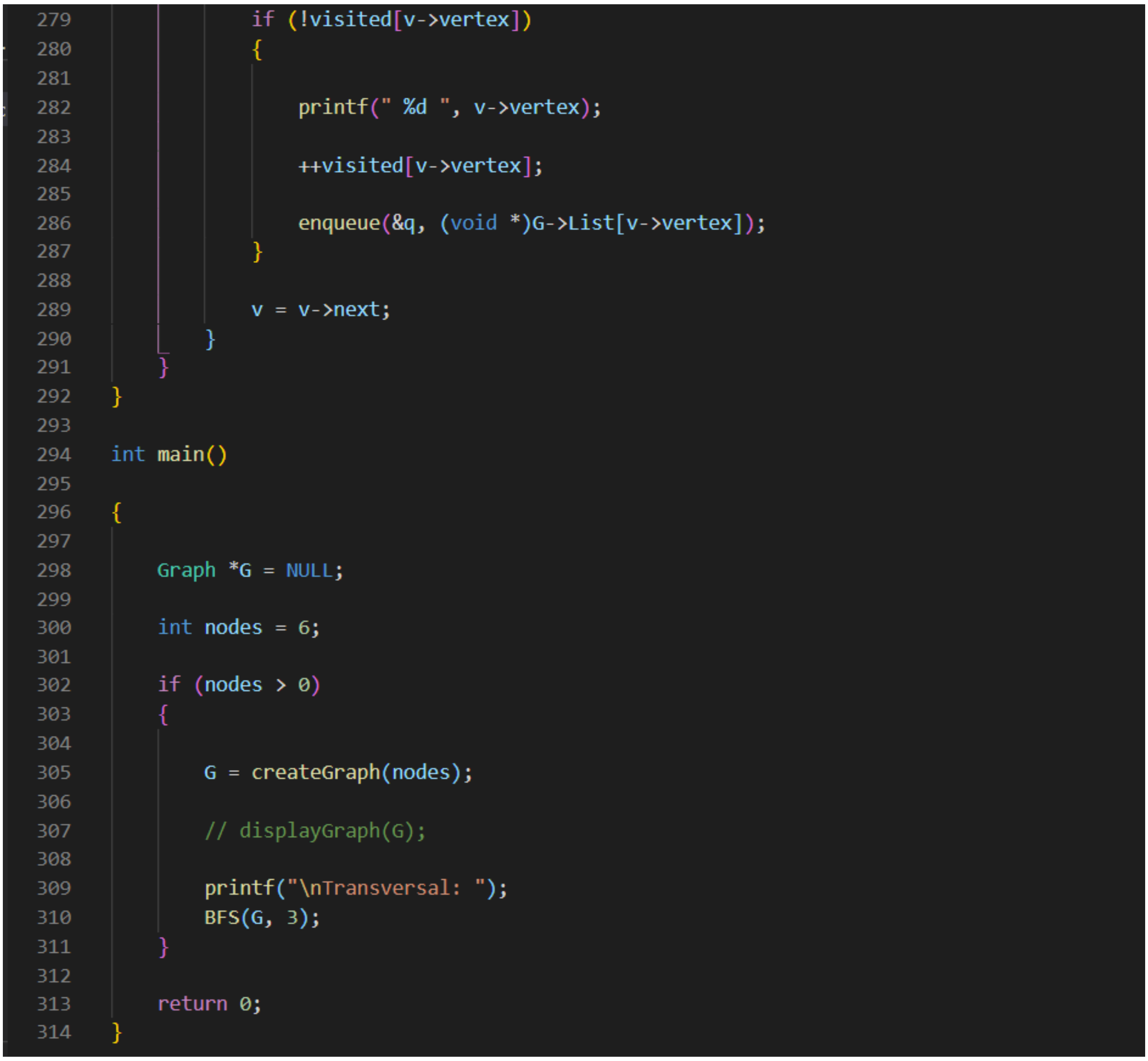
```c

        G->no_of_nodes = 0;

        G->List = (Vertex **)malloc(sizeof(Vertex *) * nodes);

        if (G->List)
        {

            for (i = 0; i < nodes; i++)
            {

                G->List[i] = NULL;

                ++G->no_of_nodes;

                for (edges = 0; edges < nodes; edges++)
                {

                    if (!search_graph[i][edges])
                        continue;

                    v = (Vertex *)malloc(sizeof(Vertex));

                    if (!v)
                        return NULL;

                    v->vertex = edges;
                    v->weight = 0;
                    v->next = NULL;

                    if (!G->List[i])
                        G->List[i] = v;

                    else
                        lastv->next = v;

                    lastv = v;
                }
            }
        }
```



```c
    }

    return G;
}

void displayGraph(Graph *G)

{

    int node;

    Vertex *v = NULL;

    if (!G || G->no_of_nodes <= 0)
        return;

    for (node = 0; node < G->no_of_nodes; ++node)
    {

        printf("\n Node(%d) \n", node);

        if (G->List[node])
        {

            v = G->List[node];

            while (v)
            {

                printf("Vertex = %d (%d)", v->vertex, v->weight);

                v = v->next;

                if (v)
                    printf(" -> ");
            }

            v = NULL;
        }
    }
```

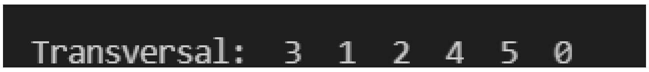


```c
}

void BFS(Graph *G, int node)

{

    Queue q = {NULL, NULL};

    Vertex *v;

    static int *visited = NULL;

    int *next;

    if (!G || !G->List)
        return;

    if (!visited)
    {

        visited = (int *)malloc(sizeof(int) * G->no_of_nodes);

        memset(visited, 0, sizeof(int) * G->no_of_nodes);
    }

    printf(" %d ", node);

    ++visited[node];

    enqueue(&q, (void *)G->List[node]);

    while (!isEmpty(&q))
    {

        v = (Vertex *)dequeue(&q);

        while (v)
        {

```



```c
            if (!visited[v->vertex])
            {

                printf(" %d ", v->vertex);

                ++visited[v->vertex];

                enqueue(&q, (void *)G->List[v->vertex]);
            }

            v = v->next;
        }
    }
}

int main()

{

    Graph *G = NULL;

    int nodes = 6;

    if (nodes > 0)
    {

        G = createGraph(nodes);

        // displayGraph(G);

        printf("\nTransversal: ");
        BFS(G, 3);
    }

    return 0;
}
```

OUTPUT:



```
Transversal:  3  1  2  4  5  0
```