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:

```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 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},
                   {0, 1, 0, 0, 1, 1, 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:

```c
/*
    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");
```

```
41          return;
42      }
43
44      temp->data = data;
45
46      temp->next = NULL;
47
48      if (!q->front)
49      {
50
51          // Empty Q
52
53          q->front = q->rear = temp;
54      }
55      else
56      {
57
58          q->rear->next = temp;
59
60          q->rear = temp;
61      }
62  }
63
64  void *dequeue(Queue *q)
65
66  {
67
68      void *data = NULL;
69
70      QNode *temp = NULL;
71
72      if (q->front)
73      {
74
75          temp = q->front;
76
77          q->front = q->front->next;
78
79          data = temp->data;
80
```

```
        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
{
```

```
120
121     int no_of_nodes;
122
123     Vertex **List;
124
125 } Graph;
126
127 int search_graph[][6] = {
128
129     {0, 1, 1, 0, 0, 0},
130
131     {1, 0, 0, 1, 0, 0},
132
133     {1, 0, 0, 1, 0, 0},
134
135     {0, 1, 1, 0, 1, 1},
136
137     {0, 0, 0, 1, 0, 0},
138
139     {0, 0, 0, 1, 0, 0}
140
141 };
142
143 Graph *createGraph(int nodes)
144
145 {
146
147     int i = 0;
148
149     int edges;
150
151     Vertex *v = NULL;
152
153     Vertex *lastv = NULL;
154
155     Graph *G = (Graph *)malloc(sizeof(Graph));
156
157     if (G && nodes > 0)
158
```

```
{

    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;
            }
        }
```

```
199         }
200     }
201 
202     return G;
203 }
204 
205 void displayGraph(Graph *G)
206 
207 {
208 
209     int node;
210 
211     Vertex *v = NULL;
212 
213     if (!G || G->no_of_nodes <= 0)
214         return;
215 
216     for (node = 0; node < G->no_of_nodes; ++node)
217     {
218 
219         printf("\n Node(%d) \n", node);
220 
221         if (G->List[node])
222         {
223 
224             v = G->List[node];
225 
226             while (v)
227             {
228 
229                 printf("Vertex = %d (%d)", v->vertex, v->weight);
230 
231                 v = v->next;
232 
233                 if (v)
234                     printf(" -> ");
235             }
236 
237             v = NULL;
```

```
    }
}

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:

```c
/*
    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");
```

```
41          return;
42      }
43
44      temp->data = data;
45
46      temp->next = NULL;
47
48      if (!q->front)
49      {
50
51          // Empty Q
52
53          q->front = q->rear = temp;
54      }
55      else
56      {
57
58          q->rear->next = temp;
59
60          q->rear = temp;
61      }
62  }
63
64  void *dequeue(Queue *q)
65
66  {
67
68      void *data = NULL;
69
70      QNode *temp = NULL;
71
72      if (q->front)
73      {
74
75          temp = q->front;
76
77          q->front = q->front->next;
78
79          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)

    {
```

```
160
161         G->no_of_nodes = 0;
162
163         G->List = (Vertex **)malloc(sizeof(Vertex *) * nodes);
164
165         if (G->List)
166         {
167
168             for (i = 0; i < nodes; i++)
169             {
170
171                 G->List[i] = NULL;
172
173                 ++G->no_of_nodes;
174
175                 for (edges = 0; edges < nodes; edges++)
176                 {
177
178                     if (!search_graph[i][edges])
179                         continue;
180
181                     v = (Vertex *)malloc(sizeof(Vertex));
182
183                     if (!v)
184                         return NULL;
185
186                     v->vertex = edges;
187                     v->weight = 0;
188                     v->next = NULL;
189
190                     if (!G->List[i])
191                         G->List[i] = v;
192
193                     else
194                         lastv->next = v;
195
196                     lastv = v;
197                 }
198             }
199         }
```

```
200     }
201
202     return G;
203 }
204
205 void displayGraph(Graph *G)
206
207 {
208
209     int node;
210
211     Vertex *v = NULL;
212
213     if (!G || G->no_of_nodes <= 0)
214         return;
215
216     for (node = 0; node < G->no_of_nodes; ++node)
217     {
218
219         printf("\n Node(%d) \n", node);
220
221         if (G->List[node])
222         {
223
224             v = G->List[node];
225
226             while (v)
227             {
228
229                 printf("Vertex = %d (%d)", v->vertex, v->weight);
230
231                 v = v->next;
232
233                 if (v)
234                     printf(" -> ");
235             }
236
237             v = NULL;
238         }
239     }
```

```
240  }
241
242  void BFS(Graph *G, int node)
243
244  {
245
246      Queue q = {NULL, NULL};
247
248      Vertex *v;
249
250      static int *visited = NULL;
251
252      int *next;
253
254      if (!G || !G->List)
255          return;
256
257      if (!visited)
258      {
259
260          visited = (int *)malloc(sizeof(int) * G->no_of_nodes);
261
262          memset(visited, 0, sizeof(int) * G->no_of_nodes);
263      }
264
265      printf(" %d ", node);
266
267      ++visited[node];
268
269      enqueue(&q, (void *)G->List[node]);
270
271      while (!isEmpty(&q))
272      {
273
274          v = (Vertex *)dequeue(&q);
275
276          while (v)
277          {
278
```

```
279                 if (!visited[v->vertex])
280                 {
281
282                     printf(" %d ", v->vertex);
283
284                     ++visited[v->vertex];
285
286                     enqueue(&q, (void *)G->List[v->vertex]);
287                 }
288
289                 v = v->next;
290             }
291         }
292     }
293
294     int main()
295
296     {
297
298         Graph *G = NULL;
299
300         int nodes = 6;
301
302         if (nodes > 0)
303         {
304
305             G = createGraph(nodes);
306
307             // displayGraph(G);
308
309             printf("\nTransversal: ");
310             BFS(G, 3);
311         }
312
313         return 0;
314     }
```

OUTPUT:

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